WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
(11 AND 33).USPT.	24
(L33 AND L11).USPT.	24

Database: US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L34

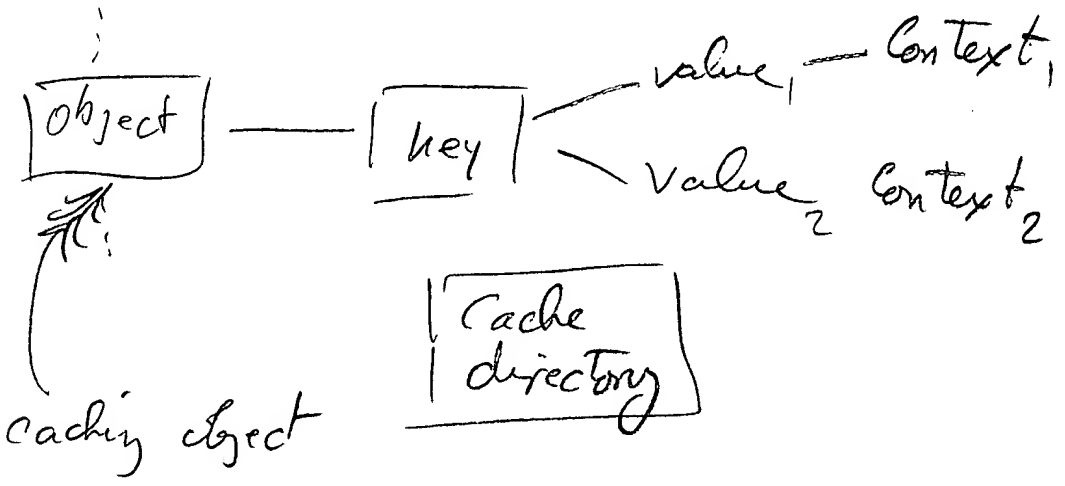
Refine Search

Recall Text | Clear

Search History

DATE: Saturday, October 04, 2003 Printable Copy Create Case

5,491,819 707/200 G06F 17/30



Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L34	L33 and l11	24	L34
L33	L32 and identify$	35	L33
L32	L31 and lock$1 and key$1	35	L32
L31	L30 and l3	76	L31
L30	caching object	123	L30
L29	L28 and l26	0	L29
L28	object$1 same key$1	22616	L28
L27	L26 and l1	0	L27
L26	L25 same l24	22	L26
L25	cach$	38058	L25
L24	multiple context	274	L24
L23	(chac$ and object$1).ab.	1	L23
L22	L21 and l12	24	L22
L21	cach$ object and different context	53	L21
L20	L18 and l12	0	L20
L19	L18 and l13	0	L19
L18	(cach$ object).ab.	48	L18
L17	L16 and l13	0	L17
L16	(cached object).ab.	16	L16
L15	L14 and l13	24	L15
L14	cached object and different context	35	L14
L13	lock$ and L12	101	L13
L12	L11 and l3	274	L12
L11	multiple context	274	L11
L10	across near5 multiple context	3	L10
L9	l1 and l2 and l3	379	L9
L8	L7 and l1 and l2 and l3	0	L8
L7	caching near3 context$1	32	L7
L6	L5 and l3 and l2 and l1	2	L6
L5	composite index	92	L5
L4	composit index	0	L4
L3	context	160056	L3
L2	cach$	38058	L2
L1	object near4 key	6145	L1

END OF SEARCH HISTORY

WEST

Generate Collection Print

L34: Entry 14 of 24 File: USPT Dec 17, 2002

DOCUMENT-IDENTIFIER: US 6496850 B1
TITLE: Clean-up of orphaned server contexts

Abstract Text (1):
A system, method and article of manufacture are provided for detecting an orphaned server context. A collection of outstanding server objects is maintained and a list of contexts is created for each of the outstanding server objects. A compilation of clients who are interested in each of the outstanding server objects are added to the list. Recorded on the list is a duration of time since the clients invoked a method accessing each of the contexts of the outstanding server objects. The list is examined at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed. Contexts that have not been accessed in the predetermined amount of time are selected and information is sent to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

Brief Summary Text (4):
The present invention relates to improving system performance and more particularly to detecting and cleaning up orphaned server contexts.

Brief Summary Text (13):
A system, method and article of manufacture are provided for detecting an orphaned server context. A collection of outstanding server objects is maintained and a list of contexts is created for each of the outstanding server objects. A compilation of clients who are interested in each of the outstanding server objects are added to the list. Recorded on the list is a duration of time since the clients invoked a method accessing each of the contexts of the outstanding server objects. The list is examined at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed. Contexts that have not been accessed in the predetermined amount of time are selected and information is sent to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

Brief Summary Text (14):
In one embodiment of the present invention, after waiting a preselected amount of time for receiving a response from one of the clients, the context may be deleted if a response from one of the clients is not received within the predetermined amount of time. In another embodiment of the present invention, a response may be received from one of the clients requesting that one of the contexts be maintained. In such an embodiment, upon receipt of the response, a time the context was last updated may be updated to a current time.

Drawing Description Text (8):
FIG. 6 is a flow diagram depicting considerations to be taken into consideration when identifying the core technologies to be used in an architecture;

Drawing Description Text (45):
FIG. 43 illustrates this Business Component Identifying Methodology including both Planning and Delivering stages;

Drawing Description Text (140):
FIG. 138 illustrates a flowchart for a method for detecting an orphaned server context in accordance with an embodiment of the present invention;

Drawing Description Text (142):
FIG. 140 illustrates a GarbageCollector requesting for interest in context A;

Drawing Description Text (143):
FIG. 141 illustrates a GarbageCollector requesting for interest in context B;

Drawing Description Text (156):
FIG. 154 illustrates a user manger/user context relationship diagram;

Drawing Description Text (197):
FIG. 195 illustrates the Context Copying Protects Context Boundaries.

Detailed Description Text (2):
A preferred embodiment of a system in accordance with the present invention is preferably practiced in the context of a personal computer such as an IBM compatible personal computer, Apple Macintosh computer or UNIX based workstation. A representative hardware environment is depicted in FIG. 1, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 110, such as a microprocessor, and a number of other units interconnected via a system bus 112. The workstation shown in FIG. 1 includes a Random Access Memory (RAM) 114, Read Only Memory (ROM) 116, an I/O adapter 118 for connecting peripheral devices such as disk storage units 120 to the bus 112, a user interface adapter 122 for connecting a keyboard 124, a mouse 126, a speaker 128, a microphone 132, and/or other user interface devices such as a touch screen (not shown) to the bus 112, communication adapter 134 for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter 136 for connecting the bus 112 to a display device 138. The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

Detailed Description Text (78):
One key challenge for today's IT managers is the need for change. Architectures provide a basic framework for major change initiatives. Clients' core business is performed by strategic applications that will most likely require frequent and rapid development to handle changes in technology capability and business requirements. A properly defined and intelligently developed architecture delivers an infrastructure on which clients can build and enhance applications that support their current and future business needs. This is how one helps clients to manage change.

Detailed Description Text (79):
A key benefit of an architecture is that it divides and conquers complexity. Simple applications benefit less from architecture than complex ones do; fewer decisions are needed in these cases, and fewer people need to know about them. During maintenance, a poorly architected small application is tolerable because it is still relatively easy to locate a fault and to anticipate the side effects of correcting it. Conversely, complex applications are more difficult to understand and to modify. Complexity is reduced by subdividing the application in layers and components, each layer having a specific functionality. The layers are strongly cohesive and de-coupled: A given layer does not need to know the internals of any other layer.

Detailed Description Text (87):
A key issue is maintainability and operability. Keep in mind that others may have to understand the rationale behind the architecture design in order to correctly maintain it.

Detailed Description Text (95):
A key goal with a delivery vehicle is that it can be reused across many applications. It is still part of the Technology Architecture, not involving application specific logic. An Application Architecture on the other hand, will be specific for a particular application.

Detailed Description Text (98):
Note: Defining a clear line between what falls under the client/server and a Netcentric technology generation is difficult; typically different people tend to have different opinions. Technologically, the Netcentric generation may be an evolution of the client/server generation. In the context of the Delivery Vehicles, the technology generation discussion may be intended to be a logical discussion that aims to highlight the new business capabilities enabled by new technologies. So for example, there could be a PowerBuilder application executing from a Web Browser using a plug-in. Whether this is called a client/server or Netcentric application is up to the reader. When presenting technology architecture information to clients, focus on the business capabilities that are offered by technologies rather than just on definitions for what is client/server or what is Netcentric technology.

Detailed Description Text (118):
During a high-level architecture design, help the user identify architecture services the user will need to address, by providing a logical level discussion one can use to assess types of base services and products needed for the specific situation.

Detailed Description Text (124):
This section should assist an architect in understanding the characteristics of, and the implications from selecting, a specific technology generation. The strengths and weaknesses of each technology generation should be understood when planning and designing a system. When identifying the core technologies to be used in an architecture, a view of the client's existing IT architecture 600, guiding principles 602 and business imperatives 604 should be taken into consideration, as depicted in FIG. 6.

Detailed Description Text (131):
However choosing a generation is not just a technical decision. Often key technology architecture decisions are made as a result of factors which are completely non-technical in nature, such as financial factors, internal and client politics (say no more), and implementation/operational considerations.

Detailed Description Text (136):
The following sections identify the main characteristics associated with a Netcentric, Client Server or Host based technology generation. This list should in no way be considered complete and exhaustive but is included as a starting point from which the identification process may begin.

Detailed Description Text (139):
The following details the importance of each of the statements in FIG. 7 and should assist one in identifying the appropriate answer for the specific client engagement.

Detailed Description Text (146):
The following section details the importance of each of the statements found in FIG. 8 and should assist one in identifying the appropriate answer for your specific client engagement.

Detailed Description Text (148):
Business Imperatives 802 B1. The application will be used only by an internal user community. Software distribution is a concern for traditional client server computing environments due to the fact that executable and data files need to reside on the client hard drive. Distribution to a user community outside of the client's organization is even more difficult to implement and manage and will probably be limited to a few key business partners. B2. The application requires an advanced, dynamic, and integrated user interface for expert users. State of the art 4GL and 3GL development languages will support advanced user interfaces which require a significant degree of context management between fields and windows. Web-based user interfaces do not support such interfaces well yet. B3. Session performance is critical to the application or sub-second response times are required for successful use. Client server applications can provide response times necessary to support transaction intensive mission critical systems. Application logic and business data can be distributed between the client and server for optimal efficiency. Web-based

interfaces still have an inherent overhead due to the connectionless communication and constant downloading of data, formatting information and applet code. B4. The application needs to support off-line mobile users. Mobile computing is becoming more prevalent in the work place, therefore, connectivity to a server can not be assumed for all user classes. A client server architecture allows for the distribution of application logic and/or data between the server and client. Replication of data and logic is usually necessary for applications that are run on portable computers.

Detailed Description Text (149):
IT Guiding Principles 804 G1. The client maintains their applications internally and the IT department has the necessary resources, organizations and processes to maintain a Client Server application. Introduction of a Client Server application to a company's production environment can require a great deal of change to the Execution, Operations and Development architectures required to develop, run and support the production systems. Before a Client Server application is developed, it is important that the client identify how a system of this type will fit within the company's strategic technology plan.

Detailed Description Text (152):
The following section details the importance of each of the statements found in FIG. 9 and should assist you in identifying the appropriate answer for your specific client engagement.

Detailed Description Text (155):
IP Guiding Principles 904 G1. The Client has the resources, organizations and processes necessary for the development and operation of a Host based application. Before a Host based application is developed, it is important that the client identify how a system of this type will fit within the company's strategic technology plan. G2. Reliance upon a single vendor (IBM) for technology solutions is acceptable. Selection of a host based architecture inherently locks the client into dependence upon one vendor for its technology solutions. While IBM is a reputable, stable company it may be important to ensure that the client's long term business strategy will be supported by IBM's technology vision and direction. G3. Centralized application and data is an acceptable strategy. A pure host based architecture eliminates the possibility of distributing data or business logic to the client. This removes some of the application performance benefits which can be seen by a distribution strategy, however, centralized access to the business logic and business data can improve operational stability and lower costs. A current trend is to transform mainframe based legacy systems into data-and application servers in a multi-tiered client/server or Netcentric architecture.

Detailed Description Text (175):
Netcentric Computing Top 10 Points Netcentric computing represents an evolution--it builds on and extends, rather than replaces, client/server. Netcentric computing has a greater impact on the entire business enterprise, hence greater opportunity and risk. Definitions of Netcentric may vary. One is about reach and content. Netcentric is not just electronic commerce; it can impact enterprises internally as well. You can begin identifying Netcentric opportunities for clients today. There are three basic types of Netcentric applications: advertise; inquiry; and fully interactive. One can underestimate the impact of Netcentric on infrastructure requirements. Build today's client/server engagements with flexibility to extend to Netcentric.

Detailed Description Text (184):
A key design decision for a client/server system is whether it should be two-tiered or multi-tiered and how business logic is distributed across the tiers. In Netcentric architectures there is a tendency to move more business logic to the server tiers, although "fatter" clients are becoming more popular with newer technologies such as Java and ActiveX.

Detailed Description Text (252):
Client/server technologies introduced new navigation metaphors. A method for allowing a user to navigate within an application is to list available functions or information by means of a menu bar with associated pull-down menus or context-sensitive pop-up menus. This method conserves screen real-estate by hiding

functions and options within menus, but for this very reason can be more difficult for first time or infrequent users. This point is important when implementing electronic commerce solutions where the target customer may use the application only once or very infrequently (e.g., purchasing auto insurance).

Detailed Description Text (257):
Windows Interaction Manager provides the application with facilities to open multiple instances of the same window. This component provides an option parameter that will let the application developers enable or disable the ability to open the same window with the same key data (that is, a duplicate instance).

Detailed Description Text (264):
Windows Interaction management allows the application to control and manage the opening and closing of multiple windows by--maintaining the parent-child relationship, controlling multiple instances of similar windows, maintaining key data-window relationship. This allows the user to work in a controlled and, well managed, environment.

Detailed Description Text (282):
HTML 4.0 and Dynamic HTML have given Web authors more control over the ways in which a Web page is displayed. But they have done little to address a growing problem in the developer community: how to access and manage data in Web documents so as to gain more control over document structure. To this end, leading Internet developers devised Extensible Markup Language (XML), a watered-down version of SGML that reduces its complexity while maintaining its flexibility. Like SGML, XML is a meta-language that allows authors to create their own customized tags to identify different types of data on their Web pages. In addition to improving document structure, these tags will make it possible to more effectively index and search for information in databases and on the Web.

Detailed Description Text (283):
XML documents consist of two parts. The first is the document itself, which contains XML tags for identifying data elements and resembles an HTML document. The second part is a DTD that defines the document structure by explaining what the tags mean and how they should be interpreted. In order to view XML documents, Web browsers and search engines will need special XML processors called "parsers." Currently, Microsoft's Internet Explorer 4.0 contains two XML parsers: a high-performance parser written in C++ and another one written in Java.

Detailed Description Text (302):
Browser Extension Services provide support for executing different types of applications from within a Browser. These applications provide functionality that extend Browser capabilities. The key Browser Extensions are:

Detailed Description Text (304):
Helper Application/Viewer--is a software program that is launched from a browser for the purpose of providing additional functionality to the browser. The key differences between a helper application or sometimes called a viewer and a plug-in are: How the program is integrated with the Web browser--unlike a plug-in, a helper application is not integrated with the Web Browser, although it is launched from a Web browser. A helper application generally runs in its own window, contrary to a plug-in which is generally integrated into a Web page. How the program is installed--like a plug-in, the user installs the helper application. However, because the helper application is not integrated with the browser, the user tends to do more work during installation specifying additional information needed by the browser to launch the helper application. How the program is initiated--the user tends to initiate the launching of the helper application, unlike a plug-in where the browser does the initiation. From where the program is executed--the same helper application can be executed from a variety of browsers without any updates to the program, unlike a plug-in which generally needs to be updated for specific browsers. However, helper applications are still operating system dependent.

Detailed Description Text (306):
ActiveX control--is also a program that can be run within a browser, from an application independent of a browser, or on its own. ActiveX controls are developed

using Microsoft standards that define how re-usable software components should be built. Within the context of a browser, ActiveX controls add functionality to Web pages. These controls can be written to add new features like dynamic charts, animation or audio.

Detailed Description Text (317):
Hyperlink--the Internet has popularized the use of underlined key words, icons and pictures that act as links to further pages. The hyperlink mechanism is not constrained to a menu, but can be used anywhere within a page or document to provide the user with navigation options. It can take a user to another location within the same document or a different document altogether, or even a different server or company for that matter. There are three types of hyperlinks:

Detailed Description Text (318):
Hypertext is very similar to the concept of Context Sensitive Help in Windows, where the reader can move from one topic to another by selecting a highlighted word or phrase.

Detailed Description Text (321):
Customized Menu--a menu bar with associated pull-down menus or context-sensitive pop-up menus. However, as mentioned earlier this method hides functions and options within menus and is difficult for infrequent users. Therefore, it is rarely used directly in HTML pages, Java applets or ActiveX controls. However, this capability might be more applicable for intranet environments where the browsers themselves need to be customized (e.g., adding custom pull-down menus within Internet Explorer) for the organizations specific business applications.

Detailed Description Text (361):
Voice response systems are used to provide prompts and responses to users through the use of phones. Voice response systems have scripted call flows which guide a caller through a series of questions. Based on the users key pad response, the voice response system can execute simple calculations, make database calls, call a mainframe legacy application or call out to a custom C routine. Leading voice response system vendors include VoiceTek and Periphonics.

Detailed Description Text (362):
Voice recognition systems are becoming more popular in conjunction with voice response systems. Users are able to speak into the phone in addition to using a keypad. Voice recognition can be extremely powerful technology in cases where a key pad entry would be limiting (e.g., date/time or location). Sophisticated voice recognition systems have been built which support speaker-independence, continuous speech and large vocabularies.

Detailed Description Text (372):
Oracle 7.3; Sybase SQL Server; Informix; IBM DB/2; Microsoft SQL Server Oracle 7.3--market leader in the Unix client/server RDBMS market, Oracle is available for a wide variety of hardware platforms including MPP machines. Oracles market position and breadth of platform support has made it the RDBMS of choice for variety of financial, accounting, human resources, and manufacturing application software packages. Informix--second in RDBMS market share after Oracle, Informix is often selected for its ability to support both large centralized databases and distributed environments with a single RDBMS product. Sybase SQL Server--third in RDBMS market share, Sybase traditionally focused upon medium-sized databases and distributed environments; it has strong architecture support for database replication and distributed transaction processing across remote sites. IBM DB2--the leader in MVS mainframe database management, IBM DB2 family of relational database products are designed to offer open, industrial strength database management for decision support, transaction processing and line of business applications. The DB2 family now spans not only IBM platforms like personal computers, AS/400 systems, RISC System/6000 hardware and IBM mainframe computers, but also non-IBM machines such as Hewlett-Packard and Sun Microsystems. Microsoft SQL Server--the latest version of a high-performance client/server relational database management system. Building on version 6.0, SQL Server 6.5 introduces key new features such as transparent distributed transactions, simplified administration, OLE-based programming interfaces, improved support for industry standards and Internet integration.

Detailed Description Text (400):
Three key considerations are: Who owns and uses the data? Replication products support one or more of the three ownership models: Primary site ownership--data is owned by one site; Dynamic site ownership--data owned by one site, however site location can change; and Shared site ownership--data ownership is shared by multiple sites. Which of the four basic types of replication style is appropriate? The four styles are: Data dissemination--portions of centrally maintained data are replicated to the appropriate remote sites; Data consolidation--data is replicated from local sites to a central site where all local site data is consolidated; Replication of logical partitions--replication of partitioned data; and Update anywhere--multiple remote sites can possible update same data at same time. What is the acceptable latency period (amount of time the primary and target data can be out of synch)? There are three basic replication styles depending on the amount of latency that is acceptable: Synchronous--real-time access for all sites (no latency); Asynchronous near real-time--short period of latency for target sites; Asynchronous batch/periodic--predetermined period of latency for all sites.

Detailed Description Text (407):
Client-server systems often require data access from multiple databases offered by different vendors. This is often due to integration of new systems with existing legacy systems. The key architectural concern is in building the application where the multi-vendor problem is transparent to the client. This provides future portability, flexibility and also makes it easier for application developers to write to a single database access interface. Achieving database access transparency requires the following: Standards Based SQL API--this approaches uses a single, standards based set of APIs to access any database, and includes the following technologies: Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), and Object Linking and Embedding (OLE DB). SQL Gateways provide a mechanism for clients to transparently access data in a variety of databases (e.g., Oracle, Sybase, DB2), by translating SQL calls written using the format and protocols of the gateway server or primary server to the format and protocols of the target database. Currently there are three contending architectures for providing gateway functions: Distributed Relational Data Access (DRDA) is a standard promoted by IBM for distributed data access between heterogeneous databases. In this case the conversion of the format and protocols occurs only once. It supports SQL89 and a subset of SQL92 standard and is built on top on APPC/APPN and TCP/IP transport stacks. IBI's EDA/SQL and the Sybase/DI Open Server use SQL to access relational and non-relational database systems. They use API/SQL or T-SQL respectively as the standard interface language. A large number of communication protocols are supported including NetBIOS, SNA, DecNET, TCP/IP. The main engine translates the client requests into specific server calls. It handles security, authentication, statistics gathering and some system management tasks.

Detailed Description Text (430):
Versioning Services maintain a historical record of the changes to a document over time. By maintaining this record, these services allow for the re-creation of a document as it looked at any given point in time during it's evolution. Additional key versioning features record who made changes when and why they were made.

Detailed Description Text (432):
Replication Services support an environment in which multiple copies of documents must be maintained. A key objective is that documents should be shareable and searchable across the entire organization. Therefore, the architecture needs to provide logically a single repository, even though the documents are physically stored in different locations. The following might be possible if documents are replicated on alternative server(s): better availability or recoverability of a distributed application; better performance; reduced network cost; etc.

Detailed Description Text (443):
Therefore, most document management products provide index services that support the following methods for searching document repositories: Attribute Search--scans short lists (attributes) of important words that are associated with a document and returns documents that match the search criteria. For example, a user may query for documents written by a specific author or created on a particular date. Attribute

search brings the capabilities of the SQL-oriented database approach to finding documents by storing in a database the values of specially identified fields within a document and a reference to the actual document itself. In order to support Attribute Search an index maintains documents' attributes, which it uses to manage, find and catalog documents. This is the least complicated approach of the searching methods. Full-text Search--searches repository contents for exact words or phrases and returns documents that match the search criteria. In order to facilitate Full-text Search, full-text indexes are constructed by scanning documents once and recording in an index file which words occur in which documents. Leading document management systems have full-text services built-in, which can be integrated directly into applications. Context Search--searches repository contents for exact words or phrases. Also, searches for related words or phrases by using synonyms and word taxonomies. For example, if the user searches for auto, the search engine should look for car, automobile, motor vehicle, etc. Boolean Search--searches repository contents for words or phases that are joined together using boolean operators (e.g., AND, OR, NOT). Same type of indexes are used for Boolean Search as for Full-Text Search.

Detailed Description Text (447):
Storage Services manage the document physical storage. Most document management products store documents as objects that include two basic data types: attributes and content. Document attributes are key fields used to identify the document, such as author name, created date, etc. Document content refers to the actual unstructured information stored within the document. Generally, the documents are stored in a repository using one of the following methods: Proprietary database--documents (attributes and contents) are stored in a proprietary database (one that the vendor has specifically developed for use with their product). Industry standard database--documents (attributes and contents) are stored in an industry standard database such as Oracle or Sybase. Attributes are stored within traditional database data types (e.g., integer, character, etc.); contents are stored in the database's BLOB (Binary Large Objects) data type. Industry standard database and file system--Documents' attributes are stored in an industry standard database, and documents' contents are usually stored in the file-system of the host operating system. Most document management products use this document storage method, because today, this approach provides the most flexibility in terms of data distribution and also allows for greater scalability.

Detailed Description Text (451):
Directory services play a key role in network architectures because of their ability to unify and manage distributed environments. Managing information about network resources involves a variety of processes ranging from simple name/address resolution to the logical integration of heterogeneous systems to create a common view of services, security, etc.

Detailed Description Text (618):
PeerLogics PIPES; IBM MQSeries; BEAs MessageQ; Momentum XIPC; Microsoft MQ (Falcon); TibCo's Rendezvous Message Passing PeerLogic's PIPES PIPES Platform applications communicate through a messaging interface that allows asynchronous, non-blocking communications. The messaging model is well-suited to complex multi-tier applications because it inherently supports asynchronous, event-driven communications. Message Queuing IBM's MQSeries New features found in version 5 include: A new Internet gateway that allows customers and partners to run mission critical business applications over an unreliable network. Enhanced message distribution carries more business information, while minimizing use of networks. Performance improvements gives message transmission at least 8 times faster than previous versions Resource Coordination ensures that data held in databases is always updated completely--or not at all, if processing cannot complete. Additional developer features include further language support for C++, Java and PL/1, and interoperability with current and previous MQSeries versions. Easier implementation because MQSeries now has the same install and use characteristics as other IBM Software Servers. BEA's MessageQ Key highlights of the MessageQ product include: High performance--up to thousands of non-recoverable messages/second; hundreds of recoverable messages/second Both synchronous, and asynchronous message delivery Broadest platform support in the industry including UNIX, Windows NT, OpenVMS, and mainframes Common Application Programming Interface (API) Publish and subscribe

(broadcasting) Microsoft Windows client product with support for DLLs (Dynamically Linked libraries), Visual Basic, and Power Builder development environments Message recovery on all BEA MessageQ clients and servers Interoperability with IBM MVS/CICS and IBM MVS/IMS Large message size--up to 4 MB--eliminates need for message partitioning Momentum's XIPC XIPC is an advanced software toolset for the development of multitasking and distributed applications. XIPC provides fault-tolerant management of guaranteed delivery and real-time message queuing, synchronization semaphores and shared memory, all of which are network-transparent. Microsoft Message Queue Server (MSMQ, formerly known as Falcon) Publish and Subscribe TibCo's Rendezvous TIB/Rendezvous' publish/subscribe technology is the foundation of TIBnet, TibCos solution for providing information delivery over intranets, extranets and the Internet. It is built upon The Information Bus.RTM. (TIB.RTM.) software, a highly scaleable messaging middleware technology based on an event-driven publish/subscribe model for information distribution. Developed and patented by TIBCO, the event-driven, publish/subscribe strategy allows content to be distributed on an event basis as it becomes available. Subscribers receive content according to topics of interest that are specified once by the subscriber, instead of repeated requests for updates. Using IP Multicast, TIBnet does not clog networks, but instead, provides for the most efficient real-time information delivery possible.

Detailed Description Text (621):
Streaming is an emerging technology. While some multimedia products use proprietary streaming mechanisms, other products incorporate standards. The following are examples of emerging standards for streaming protocols. Data streams are delivered using several protocols that are layered to assemble the necessary functionality. Real-time Streaming Protocol (RTSP)--RTSP is a draft Internet protocol for establishing and controlling on-demand delivery of real-time data. For example, clients can use RTSP to request specific media from a media server, to issue commands such as play, record and pause, and to control media delivery speed. Since RTSP simply controls media delivery, it is layered on top of other protocols, such as the following. Real-Time Transport Protocol (RTP)--Actual delivery of streaming data occurs through real-time protocols such as RTP. RTP provides end-to-end data delivery for applications transmitting real-time data over multicast or unicast network services. RTP conveys encoding, timing, and sequencing information to allow receivers to properly reconstruct the media stream. RTP is independent of the underlying transport service, but it is typically used with UDP. It may also be used with Multicast UDP, TCP/IP, or IP Multicast. Real-Time Control Protocol (RTCP)--RTP is augmented by the Real-Time Control Protocol. RTCP allows nodes to identify stream participants and communicate about the quality of data delivery.

Detailed Description Text (631):
E-Mail takes on a greater significance in the modem organization. The E-Mail system, providing it has sufficient integrity and stability, can function as a key channel through which work objects move within, and between organizations in the form of messages and electronic forms. An E-Mail server stores and forwards E-Mail messages. Although some products like Lotus Notes use proprietary protocols, the following protocols used by E-Mail Services are based on open standards:

Detailed Description Text (645):
The following are major proprietary E-mail servers used in large organizations today: Lotus Notes--platform-independent client/server mail system. Notes Mail can support over 1,500 active users per server, offering Internet integration, distributed replication and synchronization. Lotus Notes also provides integrated document libraries, workflow, calendaring and scheduling, and a cc:Mail user interface. Microsofts Exchange Server--Exchange 4.0 provides a messaging and groupware platform to support collaboration solutions on Windows machines. Microsoft Exchange 5.0 has support for all of the key Internet protocols. These include POP3 for mailbox access, SMTP for mail sending and receiving, NNTP for newsgroups and discussion forums, LDAP for directory access, HTTP and HTML for access via a web browser, and SSL for security.

Detailed Description Text (666):
What are the key business requirements?

Detailed Description Text (695):
CTI Messaging has two primary functions: Device-specific communication Manages direct communications between telephony devices and data devices Allows applications to control PBXs, key telephone systems, ISDN, analog PSTN, cellular, Centrex, etc. and supports features such as address translation, call setup, call answering, call dropping, and caller ID. Provides interface to carrier networks for call delivery and call-related messaging Message mapping Translates device-specific communication to generic API and/or message set

Detailed Description Text (726):
Encryption has two main components: the encryption algorithm, which is the series of steps that is performed to transform the original data; and the key, which is used by the algorithm in some way to encrypt the message. Typically, the algorithm is widely known, while the key is kept secret. There are several types of encryption in use today, including: Secret key cryptography--uses one key (the secret key) both to encrypt the message on one side and to decrypt the message on the other side. Public key cryptography--uses two keys, the public key and the private key. The public key and private key are mathematically related so that a message encrypted with the recipient's public key may be decrypted with the recipient's private key. Therefore, the public key can be widely published, while the private key is kept secret.

Detailed Description Text (731):
There are complex legal issues surrounding the use of encrypting in an international environment. The US government restricts what can be exported (in terms of encryption technology), and the French government defines encryption technology as a "weapon of war" with appropriate legal and regulatory restrictions. This is a key issue in international e-commerce today.

Detailed Description Text (746):
Authentication can occur through various means: Basic Authentication--requires that the Web client supply a user name and password before servicing a request. Basic Authentication does not encrypt the password in any way, and thus the password travels in the clear over the network where it could be detected with a network sniffer program or device. Basic authentication is not secure enough for banking applications or anywhere where there may be a financial incentive for someone to steal someone's account information. Basic authentication is however the easiest mechanism to setup and administer and requires no special software at the Web client. ID/Password Encryption--offers a somewhat higher level of security by requiring that the user name and password be encrypted during transit. The user name and password are transmitted as a scrambled message as part of each request because there is no persistent connection open between the Web client and the Web server. Digital Certificates or Signatures--encrypted digital keys that are issued by a third party "trusted" organization (i.e. Verisign); used to verify user's authenticity. Hardware tokens--small physical devices that may generate a one-time password or that may be inserted into a card reader for authentication purposes. Virtual tokens--typically a file on a floppy or hard drive used for authentication (e.g. Lotus Notes ID file). Biometric identification--the analysis of biological characteristics to verify individuals identify (e.g., fingerprints, voice recognition, retinal scans).

Detailed Description Text (755):
keys and certificates Kerberos--an encryption and key management protocol for third party authorization; vendors include CyberSAFE and Digital Equipment Corporation. VeriSign--a company that manages digital certificates.

Detailed Description Text (793):
The following IETF standard supports interoperability among security systems: IPSec Allows two nodes to dynamically agree on a security association based on keys, encryption, authentication algorithms, and other parameters for the connection before any communications take place; operates in the IP layer and supports TCP or UDP. IPSec will be included as part of IPng, or the next generation of IP.

Detailed Description Text (863):
TP monitors provide the ability to enqueue and dequeue requests to and from a reliable (stable storage) queue. Both the application and the administrator can

control the order of the messages (service requests) in the queue. Messages can be ordered LIFO, FIFO, time based, priority, or by some combination of these keys.

Detailed Description Text (893):
Cons of Using Tuxedo Tuxedo for basic c/s messaging is overkill. Expensive to purchase Can be complicated to develop with and administer System performance tuning requires an experienced Tuxedo administrator Uses IPC resources and therefore should not be on same machine w/other IPC products Must be understood thoroughly before design starts. If used incorrectly, can be very costly. Single threaded servers requires an upfront packaging design. Difficult to debug servers Does not work well with Pure Software products: Purify, Quantify Servers must be programmed to support client context data management Difficult to do asynch messaging in 3rd party Windows 3.x client tools (ex. PowerBuilder)

Detailed Description Text (931):
Environment Verification Services ensure functionality by monitoring, identifying and validating environment integrity prior and during program execution. (e.g., free disk space, monitor resolution, correct version). These services are invoked when an application begins processing or when a component is called. Applications can use these services to verify that the correct versions of required Execution Architecture components and other application components are available.

Detailed Description Text (958):
State Management Services enable information to be passed or shared among windows and/or Web pages and/or across programs. So lets say several fields in an application need to be passed from one window to another. In pseudo-conversational mainframe 3270 style applications passing data from one screen to another screen was done using Context Management Services that provided the ability to store information on a host computer (in this paper the term Context Management refers to storing state information on the server, not the client). Client/server architectures simplified or eliminated the need for Context Management (storing state information on the server), and created a need to store state information on the client. Typically, in traditional client/server systems this type of state management (i.e., data sharing) is done on the client machine using hidden fields, global variables, messages, files or local databases.

Detailed Description Text (959):
The popularity of the Internets HTTP protocol has revived the potential need for implementing some form of Context Management Services (storing state information on the server). The HTTP protocol is a stateless protocol. Every connection is negotiated from scratch, not just at the page level but for every element on the page. The server does not maintain a session connection with the client nor save any information between client exchanges (i.e., web page submits or requests). Each HTTP exchange is a completely independent event. Therefore, information entered into one HTML form must be saved by the associated server application somewhere where it can be accessed by subsequent programs in a conversation.

Detailed Description Text (974):
Active Help Services enable an application to provide assistance to a user for a specific task or set of tasks. Context-sensitive help is most commonly used in applications today, however this can imply more "active" support that just the F1 key. Typically, today's systems must be architected to include Help that is aware of both the user's environment, process and context, and in this sense can be called "active". Active Help services may include components like Wizards for walking a user through a new process, stored or real-time multi-media support, on-demand computer Based Training, etc.

Detailed Description Text (985):
An Application Integration Interface provides a method or gateway for passing context and control of information to an external application. The Application Integration Interface specifies how information will be passed and defines the interface by which other applications can expect to receive information. External applications in this context could include anything from Integration Performance Support systems to ERP systems like SAP or Peoplesoft to external custom applications that have been previously developed by the client.

Detailed Description Text (1024):
Report Services are facilities for simplifying the construction and delivery of reports or generated correspondence. These services help to define reports and to electronically route reports to allow for online review, printing, and/or archiving. Report Services also support the merging of application data with pre-defined templates to create letters or other printed correspondence. Report Services include: Driver Services. These services provide the control structure and framework for the reporting system. Report Definition Services. These services receive and identify the report request, perform required validation routines, and format the outputted report(s). After the request is validated, the report build function is initiated. Report Build Services. These services are responsible for collecting, processing, formatting, and writing report information (for example, data, graphics, text). Report Distribution Services. These services are responsible for printing, or otherwise distributing, the reports to users.

Detailed Description Text (1031):
The report initiation function is the interface for reporting applications into the report architecture. The client initiates a report request to the report architecture by sending a message to the report initiation function. The responsibility of report initiation is to receive, identify, and validate the request and then trigger the report build process. The main components of reporting initiation are the following. Receive, identify, and validate a report request. The identification function determines general information about the request, such as report type, requester, quantity to be printed, and requested time. Based on the report type, a table of reports is examined in order to gather additional report-specific information and perform required validation routines for the report request. After the report identification and validation functions have been successfully completed, the reporting process can continue. If any errors are identified, the report initiation function will return an error message to the requester application. Initiate report execution. The initiate report execution function processes the report profile and specific distribution requirements and determines the report to be created. It then passes control to the report execution process.

Detailed Description Text (1048):
The requester ID, report name, and date/time are used to uniquely identify the report. These values are passed to APIs which request report status, print or delete a previously generated report.

Detailed Description Text (1057):
FIG. 32 shows the module hierarchy for the custom report process. The Figure shows the relationships between modules, not their associated processing flows. It should be used to identify the calling module and the called modules for the process. FIG. 32 illustrates the Architecture Manager library 3200 which supports the report process.

Detailed Description Text (1066):
Print Report. The Print Report function sends a generated report output file to a specified or default printer. The report name and requesting process ID is passed to identify the report.

Detailed Description Text (1110):
The following are some of the architectural and integration issues that must be addressed: Process integration The workflow system must achieve a seamless integration of multiple processes. The workflow system must control the business process, eg it should be able to open a word processor with the relevant data coming from a previous business process; Infrastructure integration from PC to mainframe The ability to interface with the host-based hardware, system software, and database management systems is critical. This is essential because the workflow system is located between the client-based and host-based processes, ie it can initiate client-based as well as host-based applications; LAN and WAN connectivity Connectivity must include all sites for the supported processes, enabling a large number and variety of users to use the workflow system, and thus to execute the business process; Integration of peripherals The workflow system should support many

different types of printers, modems, fax machines, scanners, and pagers. This is especially important because of the diversity of the users that will be involved, from field crew to managers, each with their own needs and preferences; and Integration with workflow-participating applications The key to the efficiency of the workflow system is its capability to integrate with office automation, imaging, electronic mail, and legacy applications.

Detailed Description Text (1111):
Workflow can be further divided into the following components: Role management Role management ie provides for the assignment of tasks to roles which can then be mapped to individuals. A role defines responsibilities which are required in completing a business process. A business worker must be able to route documents and folders to a role, independent of the specific person, or process filling that role. For example, a request is routed to a supervisor role or to Purchasing, rather than to "Mary" or "Tom." If objects are routed to Mary and Mary leaves the company or is reassigned, a new recipient under a new condition would have to be added to an old event. Roles are also important when a number of different people have the authority to do the same work, such as claims adjusters; just assign the request to the next available person. In addition, a process or agent can assume a role; it doesn't need to be a person. Role Management Services provide this additional level of directory indirection. Route management Route management enables the routing of tasks to the next role, which can be done in the following ways: Serial--the tasks are sequentially performed; Parallel--the work is divided among different players; Conditional--routing is based upon certain conditions; and Ad hoc--work which is not part of a predefined process. Workflow routing services route "work" to the appropriate workflow queues. When an application completes processing a task, it uses these services to route the work-in-progress to the next required task or tasks and, in some cases, notify interested parties of the resulting work queue changes. The automatic movement of information and control from one workflow step to another requires work profiles that describe the task relationships for completing various business processes. The concept of Integrated Performance Support can be exhibited by providing user access to these work profiles. Such access can be solely informational--to allow the user to understand the relationship between tasks, or identify which tasks need to be completed for a particular work flow--or navigational--to allow the user to move between tasks. Route Management Services also support the routing and delivery of necessary information (e.g., documents, data, forms, applications, etc.) to the next step in the work flow as needed. Rule Management A business process workflow is typically composed of many different roles and routes. Decisions must be made as to what to route to which role, and when. Rule Management Services support the routing of workflow activities by providing the intelligence necessary to determine which routes are appropriate given the state of a given process and knowledge of the organization's workflow processing rules. Rule Management Services are typically implemented through easily maintainable tables or rule bases which define the possible flows for a business event. Queue Management These services provide access to the workflow queues which are used to schedule work. In order to perform workload analysis or to create "to do lists" for users, an application may query these queues based on various criteria (a business event, status, assigned user, etc.). In addition, manipulation services are provided to allow queue entries to be modified. Workflow services allow users and management to monitor and access workflow queue information and to invoke applications directly.

Detailed Description Text (1168):
A pattern is a named nugget of insight that conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns. Patterns are a more formal way to document codified knowledge, or rules-of-thumb.

Detailed Description Text (1171):
Patterns are usually concerned with some kind of architecture or organization of constituent parts to produce a greater whole. Richard Gabriel, author of Patterns of Software: Tales From the Software Community, provides a clear and concise definition of the term pattern: Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in

that context, and a certain spatial configuration which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant. The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when one must create it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which may generate that thing.

Detailed Description Text (1182):
Balancing tradeoffs is key to applying components for mission-critical systems

Detailed Description Text (1186):
The Management Considerations section discusses the key benefits, risks, and issues introduced by a component engagement. Key topics include: Managing risk in balancing tradeoffs between strategy, people, process, and technology Considering issues related to configuration management, testing, and performance of object systems Addressing the component development learning curve Differences between development architecture considerations leveraging the advantages of a component industry.

Detailed Description Text (1190):
Netcentric Patterns focus on how to design and leverage application frameworks, which are pieces of reusable application architecture that provide a highly configurable, flexible and maintainable system. They are aligned with SAF and/or DAF service layers. Alignment with SAF and/or DAF makes the patterns easier to grasp the context for which they are solving problems.

Detailed Description Text (1192):
For a high-level description of the context for the patterns within a service layer of SAF and/or DAF, click the title of the section. Please refer to the SAF and/or DAF for more detailed descriptions of the service layers. From the Frameworks Main Page, under Framework Extensions, the "Component Technology Extension" describes, in the context of the Netcentric Architecture framework, the additional, specialized, architecture services that are required when building a system using component technologies.

Detailed Description Text (1225):
FIG. 41 illustrates what makes up a Partitioned Business Component 4100. As long as a component does what it's suppose to do, it doesn't matter what kind of code is used to build the component's internal workings. It could be anything from COBOL to Java. This is a key benefit of encapsulation. Classifying this code is a different matter. Some code 4102 is specific to the Partitioned Business Component. Other code is more widely reusable, both functionally and technically; this is where one finds Engineering Components 4104. Another possibility is to "wrap" existing code 4106 from legacy and packaged systems. Finally, it's important to note that patterns and frameworks are frequently used as starting points for designing and building this code.

Detailed Description Text (1227):
A pattern is "an idea that has been useful in one practical context and will probably be useful in others." Think of them as blueprints, or designs for proven solutions to known problems. Having found the right pattern for a given problem, a developer must then apply it. Examples of patterns include: an analysis pattern for hierarchical relationships between organizations and/or people, a design pattern for maintaining an audit trail, a design pattern for applying different levels of security to different user types, and a design pattern for composite relationships between objects.

Detailed Description Text (1228):
A framework is a template for the implementation of a particular function (similar to a shell program). It usually embodies a known pattern (or group of patterns) in a specific technical environment. Frameworks are available from a number of third-party vendors, and they are also developed on projects. Developers are typically expected to customize and extend frameworks to meet their specific requirements, but this involves a tradeoff. Customizing and extending a framework

may optimize its use, but the resulting framework tends to be less abstract, and therefore less reusable in other contexts. Examples of frameworks include: a framework for displaying an object and its properties in Smalltalk, a Java-specific framework for persisting data, and a messaging and publish/subscribe framework for DCOM.

Detailed Description Text (1232):
Another key to embracing change is the predictability and conceptual integrity of the parts that make up an application. Fred Brooks, author of The Mythical Man-Month, writes, ". . . conceptual integrity is the most important consideration in system design." Therefore, components must be conceptually whole, and they must perform functions that are aligned with their purpose and within their sphere of knowledge. If they accurately reflect the real world, they are much easier to develop and maintain. If the real world changes, so must the corresponding component.

Detailed Description Text (1236):
Business Components model the business, and thus they enable applications to more completely satisfy the business needs. They also provide a business-oriented view of the domain and consequently a good way to scope the solution space. This results in a good context for making process and application decisions. Finally, Business Components provide a common vocabulary for the project team. They educate the team in what's important to the business.

Detailed Description Text (1238):
To manage the complexity of a large problem, it must be divided into smaller, coherent parts. Partitioned Business Components provide an excellent way to divide and conquer in a way that ties the application to the business domain. They provide the ability to "package software capabilities into more manageable (and useful) chunks." By contrast, traditional modules are too cumbersome to be reusable in multiple contexts. On the other end of the spectrum, objects are too small to effectively divide and conquer; there are simply too many of them.

Detailed Description Text (1241):
Proven processes, patterns, and frameworks offer a higher level of reuse. This is one of the key advantages because it means greater agility. These mechanisms make it possible for hundreds of developers to do things consistently and to benefit from previously captured, reusable knowledge capital.

Detailed Description Text (1242):
Business Components model the business. It sounds straightforward, but even with experience it's a challenge to identify the right components and to design them for flexibility and reuse. Flexibility and reuse are certainly more achievable with Business Components, but they are not inherent to Business Components. To accomplish these goals, as the previous examples suggest, one must understand what's happening within the enterprise and across the industry. One must work with business experts who understand the factors that will influence the current and future evolution of the business domain. This will improve one's ability to anticipate the range of possible change (i.e., to anticipate the future). The Business Component Model will be more flexible and reusable if it is challenged by scenarios that are likely to take place in the future.

Detailed Description Text (1255):
What's the best way to identify Business Components?

Detailed Description Text (1257):
The following steps describe one technique for identifying Business Components. FIG. 43 illustrates this Business Component Identifying Methodology 4300 including both Planning and Delivering stages 4302, 4304: 1. Start with entity-centric Business Components. For example, the customer is a significant entity in most business domains, therefore a Customer component may be included. A Customer Business Component would encapsulate everything an organization needs to know about its customers, including customer information (e.g., name, address, and telephone number), how to add new customers, a customer's buying habits (although this might belong in a Customer Account component), and rules for determining if a customer is

preferred. Entities themselves can be physical or conceptual. For example, customers and products are physical--you can touch them. Orders, on the other hand, are conceptual. An order represents a specific customer's demand for a product. You cannot touch that demand. 2. Look for process-centric Business Components next. Generally speaking, a process-centric Business Component controls the flow of a business process. For example, in the utility industry, a Billing component would process customer, product, pricing, and usage information into a bill. Sometimes one will find an entity associated with the process--in this case, a bill or invoice--but another option is to model this entity as a separate, entity-centric Business Component, thus decoupling it from the process.

Detailed Description Text (1258):
What's the best way to identify the responsibilities of a business component?

Detailed Description Text (1265):
From a logical perspective, components and objects are the same. They both model concepts from a particular domain, and they both encapsulate information and behavior. On this level, good component models and good object models share the same characteristics: high cohesion, low coupling, reusability, well defined services, and more. One might argue that granularity is a key difference. After all, for an object-oriented design, components are made up of objects. This may be true, but in reality both of them come in all sizes, thus making this difference rather insignificant.

Detailed Description Text (1266):
From a physical perspective, components and objects are similar, but different. The key difference relates to the different ways in which they are implemented. As long as a component's interfaces comply with an accepted standard like COM, JavaBeans, or CORBA, its internal workings can be implemented using any technology (e.g., Java, Visual Basic, Smalltalk, C, or even COBOL). The internal workings of an object, on the other hand, can only be implemented using object technology. For the same reason (i.e., standard interfaces), it is possible to request a component's services from any platform. That's not true of objects, unless they are wrapped with interfaces that comply with the accepted standards, which would make them distributed objects (i.e., components) instead.

Detailed Description Text (1279):
So one can start out building an inventory management application and then build the ready-to-reuse Inventory component which, without modification, can support many other uses. In this way one can unload the concept of inventory so that it can be reused outside the context it was initially planned for.

Detailed Description Text (1280):
This section highlights key messages for project management. The Management Lessons discuss these points further.

Detailed Description Text (1290):
To ensure that short-term concerns do not outweigh the potential benefits, project management should maintain a realistic view of the benefits and risks of components. Thus, recruiting a project champion or sponsor with a balanced, long-term view is a key to success.

Detailed Description Text (1317):
Managing Risk is Key

Detailed Description Text (1330):
The presence of a reusable business component model is a key characteristic

Detailed Description Text (1332):
The presence of a component model distinguishes component-based systems from procedural, client/server systems. In a procedural approach, there is no shared business component model. This typically requires, for example, programs to pass data to each other in a context record. Thus, any changes to the data may affect many programs. The extent of business logic reuse is also usually less with the procedural approach.

Detailed Description Text (1354):
Leveraging Expert Mentors and Time Are Key to Scaling the Learning Curve

Detailed Description Text (1358):
Thus, leveraging experienced component and object technology skills is key to success. Even a few skilled component developers can provide significant leverage to mentor and support an inexperienced development team. Experience has shown that at least 20% of the development team should have component technology or process skills at the outset. This represents a minimal level for large engagement teams with projects of one year or more duration. Smaller teams or shorter duration projects may typically require more. It is also extremely important to have a significant percentage of the team with client/server skills, to reduce additional learning curves such as GUI design or client/server architecture development.

Detailed Description Text (1394):
Leveraging Experienced Component Practitioners Is Key

Detailed Description Text (1395):
Leveraging experienced component technology skills is key to success. Even a few skilled component developers can provide significant leverage to mentor and support an inexperienced development team. At least 20% of the implementation team should have component skills Small teams or short projects likely require more

Detailed Description Text (1397):
It is important to note that component technology skills cover a wide range of competencies--from modeling and design skills to detailed programming syntax. Rarely may one individual have the necessary expertise in all these areas. Thus, experience has shown that it is necessary to find individuals that specialize in one of these areas to leverage across a large team. The key is obtaining the right balance of technology and methodology skills.

Detailed Description Text (1438):
Early Experience Has Identified Key Predictors of Success

Detailed Description Text (1440):
Key predictors of success can be drawn from this experience and others. It is important to recognize that the list below is drawn from a very small experience base. As one's experience grows, the list of traits may be refined with-hopefully-more objective measurability. This may be key to helping both a user and clients to be more successful with components.

Detailed Description Text (1455):
Thus, many engagements may need a more flexible model with training time staggered in smaller chunks. For example, the training may be accomplished through some combination of formal classroom training done in waves, self-study, case study experience with mentoring, reading, and on-the-job training. The key point, however, is that a significant commitment to training must be made-whether done upfront or spread throughout the project.

Detailed Description Text (1458):
On-going support is necessary to help developers continue building skills. On-going training is important because the entire development lifecycle is affected, to some degree, by the shift to components. An individual's first few assignments should be carefully planned to enable growing skills, and to identify people who demonstrate aptitude. Time must also be allowed for scaling the productivity learning curve, after initial skills are developed. This generally requires a fair degree of commitment from experienced frameworks developers to provide mentoring.

Detailed Description Text (1461):
A skills certification process helped to: More rigorously identify and describe competencies of what is really desired in terms of skills and competence; and, what habits should be discouraged and flagged as performance problems. Track peoples' growth-it encourages improvement by challenging people. Provide a more effective way to assign appropriate roles to people and offer up the opportunity for people to

grow into a more challenging role as quickly as they are adequately prepared. Support more effective communications of what resources had which skills (e.g., through a wallchart)

Detailed Description Text (1463):
Component-based development requires more time to scale the learning curve, because it has multiple dimensions. Component technology skills cover a wide-range of competencies including analysis, design, programming, and management. Thus, leveraging expert mentors and skills, investing in adequate training, and ensuring continued support are all key to success.

Detailed Description Text (1480):
Keep in mind, however, the benefits of this partitioning approach may be influenced by the degree with which these components interact. Thus, determining the appropriate granularity of the components is a key, strategic design decision.

Detailed Description Text (1487):
Architecture roles must be defined to support this greater degree of specialization. One engagement used the following partitioning strategy: Functional architect--responsible for resolving decisions for what the system should do. This person is ideally a user with a solid understanding of systems, or a systems person with a good understanding of, and relationship with, the users. Technology architect--responsible for delivering the platform, systems software, and middleware infrastructure to support execution, development, and operations architectures. User interface architect--responsible for setting direction of the user interface metaphor, layout standards, and integrated performance support (IPS). Application frameworks architect--responsible for designing, delivering, and supporting the application framework that provides the overall structure, or template, of the application. Object model architect--responsible for identifying and resolving modeling issues necessary to achieve a high degree of business reuse and modeling consistency.

Detailed Description Text (1489):
One must be very careful in ensuring that application frameworks are not "over-architected". Experience has shown that many frameworks fall by the way-side for the simple reason that they were not built closely enough in conjunction with the application development. They become too theoretical, complicated and over-engineered making them performance bottlenecks and obstacles to simplifying the application architecture. It has been found that frameworks should "fall out" of the application domain as candidates become obvious. Experienced developers that work closely with the application can quickly identify repetitive constructs and abstract useful frameworks from this context.

Detailed Description Text (1529):
The way roles work together is also different. For example, because of the iteration and coupling required between design and code, hand-offs from designer to programmer generally do not work well. One scenario used to leverage skills involved a lead designer creating the design, prototyping the solution, and stubbing-out methods with comments. The details were then flushed out by a junior developer. Leveraging by review and mentoring has also been key.

Detailed Description Text (1621):
Performance prototypes primarily address technology architecture questions. For example, the architecture team may need to decide early on whether to use messaging, remote procedure calls, or shipped SQL statements for distribution services between client and server. A prototype is often the only way to identify the most effective solution.

Detailed Description Text (1639):
A traditional client/server implementation often incorporates some limited iteration with a waterfall approach. This iteration is usually confined to technology architecture tasks. Component-based systems tend to require somewhat greater iteration for three key reasons: Reusability often requires actually reusing the component to ensure the reused piece meets requirements Component technology is new, thus iteration helps address greater technical risk Component skills and

methodologies are emerging, therefore the team often gains valuable experience from iteration

Detailed Description Text (1658):
On large projects, the set of components involved in a business event are often developed by many different people. Thus, the complexity of team integration further complicates the testing effort. Well-defined component boundaries in the software and organization are certainly key. However, the organization must expect the need to support flexible integration testing teams that form to ensure a particular business function works correctly across partitions.

Detailed Description Text (1674):
When developing components using objects, regression testing becomes even more important. For example, inheritance often results in sub-classes coupled to their parent. A parent class may have side effects with subtle implications to children, which are difficult to identify for test cases. Experience has shown that even seemingly innocuous changes to a parent can damage previously tested sub-classes.

Detailed Description Text (1675):
In general, an inherited feature must be retested in the context of the subclass. Retesting can only be avoided if subclasses are "pure" extensions of their superclasses; that is, if they don't override any methods and do not modify inherited instance variables. Furthermore, test cases usually cannot be inherited when overriding a method. Slight differences in logic and data declarations are indeed enough to invalidate the superclass' test cases, requiring new test definition and input data.

Detailed Description Text (1691):
This section highlights key messages for development architecture teams in regard to supporting teams and tools within a component based development project.

Detailed Description Text (1704):
Key considerations.

Detailed Description Text (1717):
The advent of client/server has focused significant attention on the importance of configuration management as key to success. Configuration management is more than just source code control. It must encompass the management of the application software components from conception, through implementation, delivery, and enhancements. While the problem is not unique to component and object development, an object-oriented environment presents special challenges discussed below.

Detailed Description Text (1722):
For example, a key principle of object-oriented design is separation of concern, which decomposes behavior into smaller, more cohesive objects. This strategy strives to prevent changes from rippling through many objects. The implication of this design approach, however, is that the resulting system may comprise many more modular pieces than a traditional architecture. This greater decomposition complicates configuration management.

Detailed Description Text (1737):
An object-oriented system must assign component ownership at multiple levels. Business process owners are still necessary; however, clear lines of responsibility must be assigned for the domain object model. Often these two may have a tight relationship. For example, consider a gas utility customer system that provides customer service orders. The service order business process and service order domain object owner should probably be the same person. However, the service order process may also need to collaborate with other key domain components such as the customer and premise. This requires collaborating and communicating with other developers.

Detailed Description Text (1759):
Regression testing is key to effective configuration management

Detailed Description Text (1776):
Tools are necessary to identify categories of similar behavior such as the class

hierarchy, where used, senders of, implementors of, etc. Today, many environments for C++, Smalltalk, Visual Basic & Java provide robust browsers with this comprehensive functionality. Additionally case tools also provide search capabilities. Unfortunately every tool uses a different method for finding artifacts, such as text searches for documents, menu provided searches in case tools, and where used and senders of within browsers.

Detailed Description Text (1781):
One key improvement in component-based development from traditional development is the use of components to assemble solutions. This is very different from libraries. Because of the reflective nature of components, runtime binaries can be dropped into the development environment, their interfaces exposed and then integrated into the current solution space. This is done through the Java Reflection mechanisms within class files and type libraries in the Active/X world.

Detailed Description Text (1785):
In the past code generation was crude, had to be customized, and was hard to keep synchronized once source code was emitted. This awkwardness was caused by other related factors like the lack of a common information model, little coupling with the IDE and no common repository sources. In addition, the ability of the CASE tool environment to comprehend the run time environment was poorly supported in most tool environments. The most damaging problem is the failure of CASE tool providers to "own" the code integration and generation produced from the model. Some of the efforts to integrate with IDE's via Add-Ins are a step in the right direction but, some key issues, such as identity integrity across multiple environments, have not yet been addressed to ensure its success.

Detailed Description Text (1791):
This desktop tool integration strategy needs to take into account the comprehensive approach used by the configuration management strategies. In other words, relevant documents need to be associated with the components and business processes they update so that key stakeholders can subscribe to alarms that may make them aware of when sections of documentation need updating. This process may help ensure that the publishing model is dynamic and current.

Detailed Description Text (1806):
Any effort to effectively address performance requires thorough measurement capabilities. There are two reasons for this. First, the team must understand where the specific risks reside, before they can effectively attack them. Is the application I/O or compute bound? Is database or network I/O a bigger issue? Are there obvious bottlenecks? These are all key questions.

Detailed Description Text (1843):
Along with good design organization, clear interface definition is key in achieving valuable tunability.

Detailed Description Text (1845):
Developers with structured programming experience often tend to perceive objects as data, manipulating them within the context of objects, effectively distributing behavior associated within components amongst all the objects which interact with it. This becomes very difficult to performance tune due to the combination of duplication of code, and the wide impact any such tuning could have on application classes. A much greater degree of performance tuning can be achieved when object responsibilities are respected and objects or collaborations of objects can be tuned in isolation with minimal impact to their embedded system.

Detailed Description Text (1852):
Opportunities for performance tuning are found both in bottlenecks and in distributed inefficiencies. There are generally many tools available in detecting bottlenecks. Distributed inefficiencies are usually more difficult to identify with tools. Whether performance optimizations are realized through cognitive analysis, or tool-assisted profiling, it is important to measure the gains against a baseline performance level.

Detailed Description Text (1874):

For some applications, an LRU caching policy might not be the right choice; a more complicated scheme with multiple cache levels might be necessary. For this reason it would be best to make the caching policy itself be an object (consider the Strategy pattern for making an object from an algorithm) so you can change the policy on demand. Cache operations and accesses. One of the reasons component-based batch performs so poorly is due to the fact that, in order to maximize modularity and preserve encapsulation, a lot of operations are performed redundantly. For instance if a balance is implemented as a calculation, and if it is needed by six different objects it is recomputed six times. These situations are very easy to identify with a performance monitor that tells you where the program spends most of its time; it is not uncommon to find that most of the time is actually spent in very few methods. For these methods (and only for them!) cache the result in an instance variable. Every time the method is invoked, check if the instance variable contains an answer: if not, compute it and store it there; if yes, just return it. Of course, each operation on the object that invalidates the result of the computation must invalidate the cache too! This technique has a very small impact on your object design and typically leaves the interface unchanged. Cache objects. Typically, this would involve leaving recently referenced objects instantiated in memory for some length of time after their last use. Then, if the object is accessed again, you check the memory-resident cache before re-loading the object from the DBMS. Usually you would construct this cache as a hash table keyed by object ID, and use a LRU policy to keep the cache size manageable. Expect degraded performance if you do anything to destroy the utility of the cache. For some applications, LRU might not be the right choice; a more complicated scheme with multiple cache levels might be necessary. For this reason it would be best to make the caching policy itself be an object (consider the Strategy pattern for making an object from an algorithm) so you can change the policy on demand. Make use of "lazy" or "deferred" loading. That is, don't do a "deep" instantiation until you know you're going to use the associated parts of the object. Instead, load selected sub-objects only when first referenced This can save on memory overhead as well as DBMS access. In some cases you can use a hybrid strategy: do a "shallow" instantiation by default, but provide the client program with a way to build the complete object on demand to provide more deterministic performance. One thing to be careful of with this approach is that if you really do tend to use most parts of the object during high-volume processing, loading it in piecemeal can actually worsen the performance, because of the overhead of maintaining the load state and because of the smaller DBMS transactions sizes. These techniques have a very small impact on your object model. De-normalize your database where possible. Typically when one does object-to-relational mappings, one tends to make every unique object type a separate table. This is best from a design perspective. But in cases where you know you have a fixed set of "private" associations (meaning physical aggregation with no possibility of shared references), then fold that sub-object data into the enclosing object's RDBMS table. It's not pretty, but it can save lots of extra loading time. Also, look at ways to do aggregate loads based on some unique object ID. For example, if you have collection-valued sub-components, insert the object ID of the enclosing object in the sub-object tables and do aggregate loads in code, rather than doing a "point-of-use" instantiation for each one separately. Of course, these optimizations can have a more substantial impact on your object model. Consider making "light" versions of some of your objects. That is, for performance critical situations, create alternate implementations of your business objects that don't have all the baggage of the first-class objects. Yes, this can be ugly and more difficult to maintain. But for many batch processing applications you might find that you can drop a lot of the (persistence-related) complexity of an object without affecting the batch processing at all. Then create fast hand-tuned routines to instantiate the "light" objects from the database.

Detailed Description Text (1884):
The primary interface to the Abstraction Factory is: "abstractType produceForKey(key)"

Detailed Description Text (1885):
where "abstractType" is the type of the common abstract interface, and key is a piece of information which identifies the appropriate concrete type. (This could be the same piece of information used in the switch/case statement; there could be a variety of ways to get it). When this method is invoked, the Abstraction Factory

consults its internal mapping and creates an "empty" object of the proper concrete class. The factory then casts the concrete object into the abstraction and returns it to the method's client. This client (a framework most likely) will then instruct the abstraction to initialize itself from the incoming data stream.

Detailed Description Text (1888):
Implementations of this pattern will vary widely depending on the selection of language. For example, in C++ a generic factory, based on templates can be constructed, and key--concrete type pairs can be registered to the appropriate instantiation of the class. This might require manual coding in other languages. The key interfaces, however are:

Detailed Description Text (1889):
Abstraction Factory AbstractType produceForKey(key)

Detailed Description Text (1915):
The key is to treat the work units as top level abstractions while they are being routed among processing nodes and to treat them as more interesting derived abstractions when internal to a node. Treating them as topmost abstractions between nodes provides a good lever for robust processing, as typical actions like IO/persistence, recovery, auditing, etc. can often be treated uniformly for all types.

Detailed Description Text (1919):
Implementations of this pattern will vary widely depending on the selection of languages and technical architectures. The key is that the all work units in the system are derived from a single abstraction. This abstraction contains key interfaces that are appropriate at the workflow level. Derived abstractions add interfaces as needed functionally.

Detailed Description Text (1927):
Compounding the difficulty in implementing the system is the fact that most batch systems must satisfy the following challenging requirements: Must be able to satisfy extremely stringent performance criteria. The system must scale to meet client's volume. The system must be flexible enough to be adapted to various contexts.

Detailed Description Text (1932):
Benefits Scalability. Each filter performs its data processing and transformation independently of other filters. By leveraging off some pipe forms' multiplexing/demultiplexing techniques, there may be several instances of a particular type of filter running in parallel. Partitioning. As a result of encapsulating each processing step within a filter component it becomes easier to manage the balance between coupling and cohesion since there are disciplined and well-defined interfaces surrounding the components. Flexibility. Since filters make little assumptions about the world around them, they can be arranged in any manner; several filters can be combined together and wrapped by a larger-grained filter; filters can be dynamically assembled at run-time depending on some context, etc.

Detailed Description Text (1945):
Common Business Components are those components from the preceding list that encapsulate key business concepts. At one level these components represent cross application components that are common to a plethora of applications. These include concepts like Customer, Company, Account, Shipment, etc. These common components normalize how basic behavior surrounding common business concepts can be normalized. Common Business Components are very concerned about the validity of the relationships they have with other components and ensuring that the information relationships are maintained correctly.

Detailed Description Text (1946):
Common Business Services deal with the higher level services that abstract out the "Business Unit of Work" or more transactional aspects of business processing. Having components that capture key processing concepts normalizes the processes for handling business events. These are services like credit checks, ordering, servicing problems, shipping, product selection, etc. They tend to capture business practices and when reused enable a company to increasingly leverage the value of those

practices.

Detailed Description Text (1949):
The patterns described in this section represent some initial attempts to capture basic concepts that are useful in the area of Common Business Facilities. They are by no means exhaustive but represent building blocks in a complete solution. Both provide tremendous value in solving two key challenges which appear on every engagement.

Detailed Description Text (1973):
The attributeValues attribute on the Attribute Dictionary is shown as an instance of the HashMap class 6004, which stores key value pairs. The HashMap Collection is used to provide access to attribute values based on the attribute name. This is required for a direct lookup of values associated with attribute names. Such lookup can use string representation of the attribute names.

Detailed Description Text (1996):
Moreover, public accessors in either strategy provide for type-safe enumerations. Enumeration is a special type of constant that deserves attention. A TypeConstant class can provide enumeration by implementing some key methods that provide for supporting iteration over the elements of the enums. In Java, for example, this entails implementing the Enumeration interface.

Detailed Description Text (2001):
Smalltalk allows for grouping logical constants in PoolDictionaries as in TextConstants. This is simply a global dictionary with key value pairs that simplifies and improves readability by using well understood names like "Space" and "Tab". However, they are global variables and they are not automatically recreated when you file in code that depends on them.

Detailed Description Text (2012):
Some languages, such as Java, support Null as a specific value, whereas other languages do not (e.g., C++ which uses zero and context to determine Null). This language mismatch can cause problems in distributed systems that use more than one language. The Null Structure pattern describes this problem and proposes a solution.

Detailed Description Text (2056):
Stateless Load Balancing. Globally Addressable Interfaces are generally implemented for stateless Servers. When Stateless Servers are used, it is a lot easier to balance the incoming load. Since state or context is always passed into the Server, any call can be directed to any Server that supports a particular operation. If one is busy, the Client can be forwarded on to the next one.

Detailed Description Text (2135):
Globally Addressable Interface 8702 services typically are used to obtain Locally Addressable Interfaces 8700 by providing some key information to the service, trigger global changes to all of the component's member objects, or to obtain component-maintained data that is not represented by a Locally Addressable Interface 8700.

Detailed Description Text (2155):
Globally Addressable Interface. The Globally Addressable Interface pattern is both a collaborating and alternative pattern. It can be used to retrieve information from Servers instead of Locally Addressable Interface--the right choice will depend on the context.

Detailed Description Text (2195):
Definitions Starting Key The Starting Key is the initial starting point for the search. The database will begin searching for data (customers in the message trace above) at the Starting Key. An example starting key could be "A*". Last Found Key The Last Found Key is used to request subsequent pages of data from the Server and the database. The "last found key" defines the starting point for the next data request. The Server will begin searching for data at the "last found key" and continue until it has retrieved a full "page" of information. When all of the data

has been retrieved from the Server and Database, the Last Found Key is left blank. This notifies the Client that all the data has been sent. Intermediate Page An intermediate "page" is returned for every request but the last. When a client receives an intermediate page and a "last found key", the client knows more "pages" of data exist on the server. In order to obtain an intermediate "page," a "last found key" must be passed from the client to the server. When the Server has retrieved a full "page" of data, the new "last found key" is saved. It is then passed back with the intermediate "page." The new "last found key" defines the starting point for the next data request. Last Page When the Server has retrieved all of the data meeting the search criteria, the Server builds the last "page." When the last page is returned to the client, the "last found key" is left blank. This notifies the client the search is complete and no more data matching the search exists on the Server. Note that the last page is usually smaller than the other pages. Empty Page When no data are selected from the search criteria, the server builds an empty page signaling to the client no more data exist on the server. Static or Dynamic Page size The page size can be defined statically or dynamically. The message trace diagram in FIG. 99 depicts a static page size. If you'd like a dynamic page size, the client must pass an additional parameter with each request to the Server. The additional parameter would be the page size. The steps associated with FIG. 99 will now be set forth.

Detailed Description Text (2196):
Collaborations 1. The user "clicks" the "Get Customers" button on the User Interface. The Client UI makes a getAllCustomers request of the Server and passes a Starting Key as a parameter. Since the user wants to view all of the customers, a Starting Key of spaces is used. Message sent=getAllCustomers(" "); 2. The Server receives the request from the Client. The Server realizes the Starting Key is blank and knows this is a new request. Thus, the Server requests first four customers (the page size) from the database. 3. The database returns the first four customers (Albert Abraham, Ned Abraham, Sally Abraham and Alice Allen) and a "Last Found Key" ("Alice Allen") to the Server. The "Last Found Key" denotes the last entry found during the search. It will be used for subsequent searches. 4. The Server builds a page with the four customers retrieved from the database. The Server returns the page and the Last Found Key to the Client. Page Type=Intermediate Page="Albert Abraham", "Ned Abraham", "Sally Abraham" & "Alice Allen" lastFoundKey="Alice Allen" The Client receives the "page" of data. The Client sends the data to a UI List Box for viewing by the user. The User can see the first two customers (Albert Abraham, Ned Abraham). The User clicks the "scroll down" arrow twice and can now see two additional customer (Sally Abraham, Alice Allen). 5. The User clicks the "scroll down" arrow again. No more data exists on the Client so the Client must request another page from the server. The Client UI makes a getAllCustomers request of the Server and passes the Last Found Key of Alice Allen. Message sent=getAllCustomers("Alice Allen"); 6. The Server receives the request from the Client. The Server requests the next four customers (page size) after Alice Allen. Message sent=getPageOfcustomer("Alice Allen") 7. The database returns the next four customers (Jason Allen, Fred Allen, Sam Allen & Zack Allen) and a "Last Found Key" ("Zack Allen") to the Server. 8. The Server builds a page with the four customers retrieved form the database. The Server returns the page and the Last Found Key to the Client. Page Type=Intermediate Page="Jason Allen", "Fred Allen", "Same Allen" & "Zack Allen" lastFoundKey="Zack Allen" The Client receives the "page" of data. The Client sends the data to a UI List Box for viewing by the user. The User can see the first two customers and one new customer (Alice Allen, Jason Allen). The User can now scroll through the next three customers. When scrolling past customer Zack Allen, the Client will request another page of data from the Server. It will follow the same basic pattern as described in steps 5-9. Eventually, the end of the list of Customer will be reached n-3. Once again, the client clicks the "scroll down" arrow and no more customers exist on the client. The Client must request another page from the server. The Client UI makes a getAllCustomers request of the Server and passes the Last Found Key of Jim Ziegler. Message sent=getAllCustomers("Jim Ziegler"); n-2. The Server receives the request from the Client. The Server requests the next four customers (page size) after Jim Ziegler. Message sent=getPageOfcustomer("Jim Ziegler") n-1. The database can only find two more customers. The database returns the final two customers (Sam Ziegler and Ziggy Ziegler) and no Last Found Key. n. The Server builds a page with the two remaining customers retrieved from the database. The Server returns the page and the blank Last Found Key to the Client.

Page Type=Last Page Page="Sam Ziegler", "Ziggy Ziegler" lastFoundKey="" The Client receives the final "page" of data. The Client sends the data to a UI List Box for viewing by the user. The User can see the following two customers (Jim Ziegler, Sam Ziegler). The User clicks the "scroll down" arrow once and can now see the final two customers (Sam Ziegler, Ziggy Ziegler) in the List Box. Subsequent "clicks" on the scroll down arrow no longer request data from the Server. The Client knows (due to the blank last found key) that it has already received all of the available data.

Detailed Description Text (2198):
Context isn't generally stored on the Server when implementing Paging Communication. As a result, it is important to request a minimum collection of data from the server. Most of the relational database are using a count mechanism that defines the maximum number of data to search. That will minimize CPU and memory usage.

Detailed Description Text (2207):
FIG. 100 illustrates a flowchart for a method 10000 for interfacing a naming service and a client with the naming service allowing access to a plurality of different sets of services from a plurality of globally addressable interfaces. In operation 10002, the naming service calls for receiving locations of the global addressable interfaces. As a result of the calls, proxies are generated based on the received locations of the global addressable interfaces in operation 10004. The proxies are received in an allocation queue where the proxies are then allocated in a proxy pool (see operations 10006 and 10008). Access to the proxies in the proxy pool is allowed for identifying the location of one of the global addressable interfaces in response to a request received from the client in operation 100010.

Detailed Description Text (2321):
Even though the "finding" and "instantiating" of a server object isn't part of this pattern, it does establish context and sets the stage for the pattern. As a result, a message trace diagram for finding and instantiating a particular object instance is shown below. This will set the stage for the implementation of the Structure Based Communication pattern.

Detailed Description Text (2394):
The rules are implemented by a different class for each type of validation. Each of these validation rule classes must know how to check its rule for every type of widget that can be checked. As mentioned in the Context section of this pattern, this will most likely be limited to text entry type widgets. In addition, each validation rule class extends an abstract validation rule class that defines what types of widgets are supported. This is an implementation of the Visitor pattern.

Detailed Description Text (2405):
Note that each of the widgets is created with a string that describes a name for the widget that the user would recognize. This name is used in the error list to help a user identify which widget failed validation.

Detailed Description Text (2437):
"Miscellaneous services" should not be interpreted as "less important services." In fact, they are vitally important. Developers are more productive when they are not required to be concerned over logging and auditing, error handling and context issues. Obtaining the freedom to largely ignore these issues requires close attention to providing facilities which are well thought out and meld into the application structure.

Detailed Description Text (2438):
Despite the pervasive demands of environmental considerations, many forms of documentation largely gloss over these issues. For example, many times when reading API documentation we find authors disclaim the fact that no error handling or "programming by contract" is shown within the code examples to improve readability. Yet, getting error handling right is key to stability in the execution environment. Programming by contract with the use of preconditions and post-conditions is perhaps the most aggressive style of programming known to date to assure correct programs. Assertion, Exception Hierarchies, Exception Response Table and Polymorphic Exception Handler tackle these problems vigorously by helping to define clearly how to solve some of these key kernel application architecture considerations. The Exception

patterns provide a blueprint illustrating how architectural issues can be abstracted into a service level component so that the impact to the application code is minimal.

Detailed Description Text (2439):
A demanding issue in distributed systems is gathering and using trusted information about the clients interacting with the systems. In earlier generations of systems the number of users was a fairly precise calculation--just count the number of workstations which could potentially connect to an application. Information about the users was also a fairly simple matter since they connected directly to the resources from which they were requesting services. Today, with clients offering web services within n-tiered architectures this is no longer easily predictable. In addition, requirements to support these less predictable numbers of users and to have a personal "one-to-one" relationship with them is key to many web strategies. The LoadBalancer and UserContext pattern offer some help in this area. The former addresses strategies for ensuring maximal leverage of the system resources and services and the latter helps in addressing the issue of maintaining state and context about the user. These facilities are mandatory when security, auditing and logging are considered essential properties of the environment.

Detailed Description Text (2443):
Each assertion may be raised with descriptions for helping to identify where the assertion failed. Also, each assertion may be raised with parameters for helping to identify why the assertion failed. In one embodiment, two types of assertion classes may be provided. In such an embodiment, one of the assertion classes may implement assertion-checking logic and the other assertion class may implement only null operations, with one of the assertion classes being selected to be raised.

Detailed Description Text (2456):
Assertions can be raised with descriptions and parameters. A description can help to identify where the Assertion failed and a parameter list can help to identify why the Assertion failed.

Detailed Description Text (2461):
Benefits Ease of Error Identification. Many error are caused by invoking an operation with improper data (parameters). By formalizing these conditions, it is very obvious is an error was caused by bad data or bad code. Correctness. Properly placed assertions assure that the system is in a correct state and responses can be trusted. Assertion checking complements, but does not replace, a comprehensive testing program. The responsibility remains with the designer to identify the correct conditions to assert. Consistency. All checks will be made and handled in a similar fashion. Control. The enabling and disabling features of the Assertion allows an operations controller to determine when and what checks should be made at runtime rather then development time. Flexibility. All handling and clean-up of incorrect assertions is located in one place making changes to this logic much easier to implement. Readability. Polices concerning how assertions are actually thrown and handled is not in the functional code. Documentation. The code actually documents the design assumptions. This can also be used by documentation generators which read through the code.

Detailed Description Text (2468):
FIG. 138 illustrates a flowchart for a method 13800 for detecting an orphaned server context. A collection of outstanding server objects is maintained and a list of contexts is created for each of the outstanding server objects in operations 13802 and 13804. A compilation of clients who are interested in each of the outstanding server objects are added to the list in operation 13806. Recorded on the list in operation 13808 is a duration of time since the clients invoked a method accessing each of the contexts of the outstanding server objects. The list is examined at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed in operation 13810. Contexts that have not been accessed in the predetermined amount of time are selected in operation 13812 and information is sent to the clients identifying the contexts that have not been accessed in the predetermined amount of time in operation 13814.

Detailed Description Text (2469):

After waiting a preselected amount of time for receiving a response from one of the clients, the context may optionally be deleted if a response from one of the clients is not received within the predetermined amount of time. Also, a response may optionally be received from one of the clients requesting that one of the contexts be maintained. In such a situation, upon receipt of the response, a time the context was last updated may be updated to a current time.

Detailed Description Text (2472):
In the design of a stateful server, the LUW Context pattern facilitates the server process constructing domain objects at the request of the clients and maintaining these objects within a given context. Domain objects are entered into a registry with their appropriate context which the server maintains and updates when a request is received to create or delete an object. Each time a context is accessed then a notification is broadcast to the registry, regardless of a state change. With a simple context management, each time a context is referenced by a client a reference counter is incremented and similarly decrements when the reference is destroyed. Once the reference count returns to 0 then the context can be removed from the registry.

Detailed Description Text (2473):
If the context is not explicitly deleted by the client then it will remain in the registry as the server has no way of detecting that the context is orphaned.

Detailed Description Text (2474):
Even if the client application is rigorously designed to ensure all redundant contexts are deleted, an abnormal client event may result in its termination leaving an orphaned server context.

Detailed Description Text (2477):
Therefore, Distributed Garbage Collection should be implemented to ensure that orphaned server contexts are deleted on the server. In the registry for the Garbage Collection the server maintains a collection of outstanding server objects and for each object a list of its contexts, the clients currently interested and the duration since a method was invoked upon a given context by a client. Periodically this list is examined to establish if any of the objects have not been accessed for some configurable time and are candidates for reaping. So, for example, a value of 5 minutes could serve as a default poll event or keep alive interval. If a candidate for a orphaned server process is identified then the clients are sent a message, requesting if they are still interested in the context. This might be performed by publishing an "is anyone interested" message to the registered clients to establish if anyone is interested in the object in its assigned context or by asking the clients explicitly depending on the nature of the architecture.

Detailed Description Text (2479):
FIG. 140 illustrates a GarbageCollector 14000 requesting for interest in context A 14002. No responses are received from any clients so the server assumes it is orphaned and deletes it.

Detailed Description Text (2480):
If the period configured for a client to respond expires then the context is deleted. This accounts not only for an abnormal termination of the client but for failure of the client application to clean up. However, if a request is received from a client to maintain a context then the time the context was last accessed is updated to the current time and it remains in the Garbage Collection registry.

Detailed Description Text (2481):
FIG. 141 illustrates a GarbageCollector 14100 requesting for interest in context B 14102. Client 2 registers interest so the reaper updates the access time stamp and maintains B.

Detailed Description Text (2482):
Benefits Cleanup on the Server. Reduces the amount of redundant resources on the server to a minimum. This is especially important if a stateful component is held in a transaction by a client and the architecture prevents additional clients from accessing it, e.g. with BEA's M3. Performance. Ensures that only the required

contexts are maintained on the server, minimizing the work that the server is required to do, especially during the cleanup process at the end of a LUW. Centralization. The collector has a central view over all of the contexts that are currently accessed by all of the clients within a given context. This simplifies the persistence of a context at the end of processing.

Detailed Description Text (2483):
In order to prevent potential race conditions the client must be given sufficient time to respond to the keep alive message from the server before the context is deleted. Typically the client has a separate listener for upward messages originating at the server, so queuing is not an issue at the client end. However, a server is more likely to queue on the receiving end, especially in a system with high message rates.

Detailed Description Text (2486):
Context Pattern Language describes the architecture that is required before the Distributed Garbage Collection is required.

Detailed Description Text (2491):
FIG. 142 illustrates a flowchart for a method 14200 for creating a common interface for exception handling. Naming conventions of exceptions are determined in operation 14202. A prefix and/or a suffix is added to each exception interface name in operation 14204 for indicating that the exception interface is an exception. In operations 14206 and 14208, where an exception error occurred is indicated and a determination is made as to what caused the exception error. Context is provided as to what was happening when the exception error occurred in operation 14210. Streaming of the exception is allowed to a common interface in operation 14212. An error message is outputted indicating that an exception error has occurred in operation 14214.

Detailed Description Text (2498):
Let's take another scenario. Suppose you want to prevent any raised exception from bringing down your system, as least not without a fight. In some cases the error will be unrecoverable and there is not much you can do but release resources (locks, communication channels, . . .) and terminate. What caused the problem is going to be on the tops of the minds of the production support people, and yours when you get their call (always in the middle of the night). You could write the exception handling logic chunks for each exception type--remembering that each exception has its own interface and will require separate logic to handle each interface--for each exception, but now you have to handle all the exceptions in the system. Wouldn't it be nice to write one chunk of handling logic and be done with it?

Detailed Description Text (2500):
The first step is to create an exception interface that all other interfaces will use or extend. It is not possible to provide one here as it greatly depends on the requirements at hand. But here are some guidelines: Determine the exception naming conventions. Use either a prefix or suffix to indicate that the interface is an exception. Also consider naming exceptions with the layer or domain they originate from. For example you may have an exception, CaAddressExcp, which is owned by the Customer Acquisition domain. Provide a means to determine where the error occurred (file, line, client or server, layer, . . .) so that it can be investigated. Provide a means to determine what happened (could not open file: XYZ). Provide context as to what was happening (Saving account information). Provide a way to stream the exception or stringify it. Consider separate production messages versus debug messages. Don't try to indicate severity. This is determined by the context of the caller, not the callee.

Detailed Description Text (2514):
FIG. 145 illustrates a flowchart for a method 14500 for recording exception handling requirements for maintaining a consistent error handling approach. An exception response table is provided in which an exception is recorded in operations 14502 and 14504. The context of the exception is entered in the exception response table in operation 14506 and a response for the exception is listed in the exception response table in operation 14508. The response is subsequently outputted upon the exception occurring in the context in operation 14510.

Detailed Description Text (2515):
A typical response and a last resort response may be listed in the exception response table. The typical response may also be outputted upon the exception occurring in the context. The last resort response may be outputted upon the exception occurring out of the context. Additionally, abbreviations may be used to reduce an output size of the exception response table. Further, the exception response table may also include an exception category field for permitting organizing multiple exceptions by source. Optionally, an optimization may be determined that can be made based on similar entries in the exception response table. Further, the optimization made may also include classifying the exceptions for organizational purposes.

Detailed Description Text (2516):
The response to an exception may vary per exception type and the context in which it is thrown, such as being thrown on the client or server, and the context in which it is handled. How do you record the exception handling requirements?

Detailed Description Text (2517):
During exception handling design there are several aspects to capture to achieve a consistent approach: The set of exceptions to be handled The set of responses to these exceptions The context in which the exception is handled; e.g. client or server, batch or GUI

Detailed Description Text (2518):
The set of exceptions to handle and their organization structure varies by project. Typically exceptions are organized into hierarchies to facilitate handling. The response to an exception may vary by exception type, the context in which it was thrown, and the context in which is handled. Here are some examples of error handling decisions of a hypothetical project: "All exceptions thrown on the server, and not handled by the server logic, will be propagated to the client." "The current transaction is aborted if a server exception is not recoverable" "All Server exceptions derived from Excp will be logged if not handled by the server code. The last resort handler will ensure this." "GUI clients will display the error information in a splitter window" "Batch clients will send error information to Operations"

Detailed Description Text (2519):
These few examples demonstrate how context (Batch, GUI, Client, Server, last resort) can affect the handling of exceptions, and that even in a given context, the exception type may play a role in the handling. In a real system there may be several other context and exception-type specific requirements.

Detailed Description Text (2520):
There are two common exception handling contexts that should be present in most systems. One is referred to as the Typical Response and the other is referred to as the Last Resort Response. The Typical Response is the error handling code intentionally added to handle exceptions. For example, car.start( ) is likely to fail due to being out of gas. The Typical Response may be to fill the tank and retry. The Last Resort Response is what to do when an exception is not handled (the Typical Response could not handle the error, such as a hole in the gas tank). Last Resort Response is a way of capturing what should be done when application code fails to handle an error. Recovery is usually not possible at this point but the handler may be coded to log the error and notify Operations of the problem. Without this response, systems may crash unnecessarily, or without indicating what happened.

Detailed Description Text (2521):
All these permutations of exception types, contexts, and responses need to be managed in order to maintain a consistent error handling approach.

Detailed Description Text (2522):
Therefore, use an Exception Response Table to capture the exceptions in the system, and the appropriate responses by context. What is important to capture is the exception, context, response, information; documenting the error handling

requirements.

Detailed Description Text (2523):
The following table lists exceptions by category and type, with the typical and last resort response. Other contexts and responses are listed within these columns. The exception category field is optional but can help to organize exceptions by their source (application, architecture, . . .) or hierarchy. This table can become quite packed with response information so a nomenclature may need to be developed to condense the information. The implementation section provides an example of this; Other ways of formatting this information are possible.

Detailed Description Text (2526):
Benefits Requirements Traceability. Exceptions requirements are captured and managed through implementation. Hierarchy Design. Analysis may show optimizations that can be made such as handling a subtree of exceptions with the same code, as the response is the same to any exception in the subtree. Interface Design. Discovery of interface requirements on the exception classes to support a particular response is another benefit. Handler design. Assists in exception handling design by identifying common responses that can be leveraged by the handlers.

Detailed Description Text (2530):
Context: C=Client S=Server

Detailed Description Text (2531):
Response: N/A=not applicable; don't handle L=log error L(diagnostic)=log errors for diagnostic purposes only N=notify operations Optional=application, context dependent. Not required to be caught P=pass exception to client P(<exception>)=pass given exception type to client, will be different from type caught Popup(wam)=display warning message Popup(severe)=display severe warning message Popup(retry)=display retry message Shutdown=release resources and shutdown gracefully.

Detailed Description Text (2567):
User Context

Detailed Description Text (2568):
FIG. 152 illustrates a flowchart for a method 15200 for maintaining a security profile throughout nested service invocations on distributed components. In operation 15202, interconnections are provided between distributed components each having nested service invocations. A user is identified in operation 15204. The user is associated with roles in operation 15206. In operation 15208, a user context instance is created upon successful identification of the user. The user context instance also includes information about the user including the roles. A request is received from the user to invoke a service on a component in operation 15210. The component invokes an additional service of another component. The user context is queried for the information about the user in operation 15212. The user information is compared with an access control list for verifying that the user has access to the component in operation 15214. The user information is also compared with an access control list for verifying that the user has access to the additional service of the other component in operation 15216.

Detailed Description Text (2569):
Optionally, all user interactions may be logged as well. As another option, a user interface may be modified to provide access to actions that can be performed by the user based on an identity of the user and the roles associated with the user. The user context instance may also be passed along as a parameter of service invocations. Additionally, the service invoked may associate any objects created, updated, or deleted with the user context instance. As a further option, the user context instance may also encapsulate security certificates of the user.

Detailed Description Text (2574):
FIG. 153 illustrates a component interaction diagram showing an interaction between a number of components in a financial system. A user initiates an addstocko service on the Portfolio component 15300. To perform the addstock( ) service, the Portfolio must use the getStockPrice( ) and the deductFromAccount( ) services on the Market

and Finance components 15302,15304, respectively. This implies that a user who can access the addStock( ) service must also have permissions to access the getStockPrice( ) and the deductFromAccount( ) services. This may need to be checked by each of the distributed components within the context of one logical service. In addition, auditing what has been done, or perhaps requested to be done, adds another common requirement that must be accounted for. A component servicing multiple clients must associate client requests with corresponding services invoked on business objects. This information must be persisted as each change is committed.

Detailed Description Text (2575):
Therefore, represent information about a user in a shared User Context object. This object maintains a user's unique identification that can be subsequently checked against a resource's access control list (ACL). A User Context instance is created upon a user's successful, validated identification to the system (usually through some "login" mechanism). After that, the system user interface can modify itself to provide only the actions that can be performed by that particular user acting in a particular role. Controls may query the User Context and modify their own visual state as needed (enable/disable, hide/show).

Detailed Description Text (2576):
The User Context can also be passed along as a parameter of service invocations. All public, stateless services on a component should provide for a User Context to be passed along as a parameter. The service being invoked can then associate any Business Objects created, updated, or deleted as a result of the service invocation with the User Context.

Detailed Description Text (2577):
One example of this would be a User Manager 15400 associating a User Context instance 15402 with the Business Objects 15404 they are affecting. FIG. 154 illustrates a user manger/user context relationship diagram.

Detailed Description Text (2579):
Benefits Common User Representation. One single representation of a user and their access rights can be shared across all areas of the system. Extensible Security. Because there is one source for the User Context various policies or strategies could be used to identity and authenticate the User within a context. For example, it could encapsulate the User's certificates that allow more advanced security strategies to determine authorization. Class UserContext UserContext(Identifier identifier) Identifier getIdentifier( ) String getname( ) void setName(String newName) void addRight(String accessArea, AccessLevel level) void removeRight(String accessArea, AccessLevel level) Vector getRights(String accessArea) boolean canCreateIn(String accessArea) boolean canReadIn(String accessArea) boolean canUpdateIn(String accessArea) boolean canDeleteIn(String accessArea) Class AccessLevel static AccessLevel create( ) static AccessLevel read( ) static AccessLevel update( ) static AccessLevel delete( ) boolean=(AccessLevel anAccessLevel)

Detailed Description Text (2580):
It is expected that the User Context will be passed from component to component. In this case the User Context will have to be defined using some sort of interface language definition (IDL).

Detailed Description Text (2587):
MTS & EJB offer an environment that does not require the passing of the context with every operation. A container as a set<context type> that provides a handle within the component for the methods to access the cached context.

Detailed Description Text (2592):
A key issue frequently encountered in the development of object-oriented systems is the mapping of objects in memory to data structures in persistent storage. When the persistent storage is an object-oriented database, this mapping is quite straightforward, being largely taken care of by the database management system.

Detailed Description Text (2594):
The impedance mismatch is due to the following contrasting features of

objects/classes and tables: Identity: Objects have unique identity, regardless of their attributes. Tables rely on the notion of primary key to distinguish rows. While a relational DBMS guarantees uniqueness of rows with respect to primary keys for data stored in the database, the same is not true for data in memory. Inheritance: This is a meaningful and important notion for classes; it is not meaningful for tables in traditional RDBMSs. Navigation: The natural way to access and perform functions on objects is navigational, i.e., it entails following references from objects to other related objects. By contrast, relational databases naturally support associative access, i.e., queries on row attributes and the use of table joins.

Detailed Description Text (2596):
A key objective of a comprehensive object-to-relational persistence architecture is shielding the application business logic and developers from the relational structure. The benefits are a simplified environment for business developers, reduced distraction with technical issues, and increased focus on the business object model and functional logic. However, in order to reap these benefits, a significant investment in architecture development is typically required.

Detailed Description Text (2599):
The patterns in this section solve several of the fundamental problems encountered in the development of an object-to-relational persistence architecture, including the mapping of classes to tables (Data Handler, Individual Persistence), identity management (Object Identifiers as Object), caching(Object Identity Cache), allocation of responsibilities (Data Handler, Piecemeal Retrieval, Persistent State Separate from Persistent Object), and data access optimization (Multi-Object Fetch) and the mapping of basic SQL types to object attributes (Attribute Converter).

Detailed Description Text (2645):
A Mapper for a class contains the columns(s) and table that the class will be written to, the type of the data and the order that they will be read from/written to the stream. It also reads and writes rows of data from the database. It generates a where clause from the primary key information (PID). A database runtime context is obtained from the Transaction Service (using the current implicit transaction context). It also contains the code to query for sequence ids, for classes that use optimistic locking.

Detailed Description Text (2657):
Create a data access architecture that supports reusable, independent business objects in the context of atomic, functionally-specific transactions.

Detailed Description Text (2711):
Therefore, implement the associations using object identifiers that contain the necessary information to retrieve the object if it is needed. These objects can then be loaded when the object is restored, eliminating the need to restore the entire associated object In addition, since these object identifiers uniquely identify an object instance, they can be used/passed in place of memory pointers. When the object is needed, simply restore the instance using the object identifier.

Detailed Description Text (2713):
The object identifier (or OID) must contain enough information to uniquely identify the instance. This identifier could be a unique row id generated by a database, a UUID generated by a utility or a unique string generated from one or more attributes. It is generally desirable to have a different class of OID for each type of object, thereby creating a more type-safe environment. It should also be noted that OID's should have value semantics.

Detailed Description Text (2717):
Identity Manager--Uses Object Identifiers as unique key's for storing persistent state objects.

Detailed Description Text (2723):
Within a client context (e.g., a logical unit of work), the same object may be referenced more than once. How can object identity be preserved and redundant accesses to persistent store be avoided?

Detailed Description Text (2733):
A dictionary can be used to implement the Object Identity Cache. The following points should be considering when implementing an Object Identify Cache.

Detailed Description Text (2744):
Context Management

Detailed Description Text (2745):
Each Context (e.g., transaction, thread, etc.) owns an Identity cache which holds all of the objects in that context.

Detailed Description Text (2749):
Optionally, the state class may support data structures of arbitrary shapes. Supporting classes may manipulate the state in a polymorphic fashion. As another option, the state may be further implemented as a class that contains key-value attribute pairs. The state class may also contain a keyed data structure containing attribute names and attribute values. Additionally, the state can also be asked to write data to a stream.

Detailed Description Text (2762):
Implement the state as a class that contains key-value attribute pairs.

Detailed Description Text (2763):
This is an alternative approach to the one listed above. Using this technique the state class would contain a keyed data structure (e.g. a dictionary) containing keys (the attribute name) and values (the attribute value). In cases where you want to copy the state or pass it to another process, the supporting code does not need to know the type of the state object it is working with. State objects can simply be asked to read or write their data to and from a stream or string. While this offers a more dynamic solution, it should be noted that with this solution additional logic would need to be included in the persistent object to insure the validity of the associated state class.

Detailed Description Text (2766):
Context Manager: Used in conjunction with Identity Manager to maintain separate collections of persistent state objects for separate application contexts.

Detailed Description Text (2801):
In the process of managing its business model, a LUW will often have to send messages to all business objects within the LUW. Examples of such messages include saveDataChanges, retrieve, or is Dirty. Rather than hardcoding a call to each object in the model, the pattern LUW Context suggests using a bag (or collection) to hold all objects referenced by the LUW. Then, a single message can be sent to the bag which will forward it to all objects within it.

Detailed Description Text (2805):
Another problem that may arise when multiple requests are sent for a given transaction is deadlock. Deadlock occurs when two requests are trying to lock the same pair of objects. Each request locks one object and waits to commit until it can lock the other. Therefore, each request will wait for the other to complete while neither is able to do so. The Request Sorter pattern works with Request Batcher to handle this problem by sorting requests as they are being unbatched by Information Services. A request is not allowed to proceed until any dependent requests are completed.

Detailed Description Text (2810):
The dependent request may not have a primary key. In such situation, the response to the parent request may include the primary key for allowing the dependent request to be responded to. As another option, the dependent request may also include a pointer indicating that the dependent request is dependent on the parent request. In this situation, the pointer may be passed to the parent request during the step of batching the requests into the network message.

Detailed Description Text (2819):

Therefore, additional mechanisms should allow a batched request to indicate that it depends on another request. If a business object does not have its primary key--or other attributes guaranteeing a unique match--it becomes a Dependent Request. Because a dependent cannot fetch itself, some other business object will inevitably have the necessary foreign key. The dependent request will therefore register its dependency on this other object.

Detailed Description Text (2827):
LUW Context

Detailed Description Text (2845):
Thus, the scope of a bag is an LUW. In addition, a bag provides contextual information for the LUW--i.e., which business objects that LUW uses. The architecture bag therefore models the LUW Context, and will be named as such.

Detailed Description Text (2846):
Benefits Encapsulation. LUW Context hides technical details of persistence, garbage collection, etc., from business developers. Some of the Known Uses have managed to hide this framework, in entirety, from business logic. Robustness. This approach guarantees that each business object in the LUW receives forwarded messages. There is no longer the chance of a developer forgetting to include a particular business object in a group message. Application Maintainability. As the application requirements change, the set of business objects in an LUW can change without impacting the generic, LUW code. For example, a future version of the Account Payment window could also display Address information. This introduces a new business object into the LUW. Yet it would not require updating saveDataChanges( ), or release( ) methods, as it would have previously. Performance. LUW Context can dramatically improve performance in a distributed environment. By nature, it batches up messages for a group. This can reduce network messaging. For example, consider a search window which has instantiated 30 business objects. Releasing those objects, if the messages were sent independently, would require 30 network messages. However, with LUW Context, a single message can go from the client to the server. Then, within the server executable, the LUW Context forwards release( ) to the 30 member objects. This is far less costly than using the network for that messaging. Because of this message batching, some readers may confuse LUW Context with Request Batcher. It is true that both reduce the number of network messages. However, the former is concerned with supporting a family of generic, architecture messages, like isDirty( ) and refresh( ), on a single atomic object. The latter is concerned with grouping database requests into a physical package, for un-batching at the server. Although both have similar principles and characteristics, they solve different problems and are implemented differently. Architecture Extensibility. LUW Context models the LUW as an actual object in the software. Any other architecture processing which executes on a per-LUW basis can also be coded there. (See the Related Patterns section for examples.)

Detailed Description Text (2847):
This pattern seeks to hide the message propagation from business logic. In fact, messaging to an LUW Context can be hidden completely in an architecture superclass. Previously, saveDataChanges( ) would've been coded specifically in each concrete activity class. LUW Context allows it to be abstracted, as in:
AbstractLUWActivity::saveDataChanges( ) { // Propagate along the save message to all business // objects in my LUW. Subclasses don't even need to // know about this method. this.getLUWContext( ).saveDataChanges( ); }

Detailed Description Text (2848):
This assumes that business objects were put into the LUW Context in the first place. The context object can be passed in when instantiating an object, transparently by the persistence and streaming frameworks, etc.

Detailed Description Text (2849):
An LUW Context can collaborate with a Request Batcher, if requests are batched for transmission to the data store. Rather than storing the batcher globally, each context and hence model can have its own manager. This allows multiple domain models, in multiple contexts, to send transactions simultaneously but independently. Then, whenever a business object requests an access or update, its request will be

intercepted by the model's particular Request Batcher. The batcher then holds these requests until the activity--which owns the LUW--tells the batcher to send them.

Detailed Description Text (2850):
The LUW Context holds onto all domain objects in a particular model. It can therefore collaborate with an Identity Registration Manager, to enforce object uniqueness within the particular context.

Detailed Description Text (2851):
The Potential Variables pattern, which provides local undo, is discussed in the first version of the Object Solutions Handbook. If local LUWs use this approach, then LUW Context is a natural location to store the LUW phase variable.

Detailed Description Text (2852):
In that pattern, every time a business object sets an attribute, the variable must be checked. The LUW Context, as intermediary, can provide a simple public interface which supports setting and querying the phase variable.

Detailed Description Text (2871):
Referential Integrity (RI) ensures that references between two relational tables are valid. That is, foreign keys in one table must refer to existing primary keys in another table. For example, RI rules could require that all accounts have a customer. Then, values in account.cust_id would need matching values in customer.cust_id.

Detailed Description Text (2872):
Mission-critical RDBMSs can enforce RI at run-time. Then, if a modified foreign key does not match an existing primary key, the database prevents the update.

Detailed Description Text (2876):
Imagine a transaction A orders customer before account. Conversely, a concurrent transaction B orders account before customer. A will request a lock on the customer table, while B will request a lock on the account table. A must wait for B to complete and release its account lock. Yet B cannot complete until A releases its customer lock.

Detailed Description Text (2893):
FIG. 191 illustrates a flowchart for a method 19100 for assigning independent copies of business data to concurrent logical units of work for helping prevent the logical units of work from interrfering with each other. In operation 19102, multiple logical units of work operating concurrently are provided. Each of the logical units of work manipulate at least one common business object. In operation 19104, a copy of the common business object is created for each of the logical units of work such that the copy of the business object for one logical unit of work becomes a separate instance from the copy of the business object for another logical unit of work. Each copy of the business object knows the context of that copy of the business object in relation to the associated logical unit of work. Upon receiving a request to make changes to a copy of the business object of one of the logical units of work in operation 19106, that particular copy of the business object is changed while the other copies of the business object are not changed. It is then verified in operation 19108 that only one copy of the business object has been changed and the common business object is updated in operation 19110 based on the change to the copy of the business object.

Detailed Description Text (2894):
A business object may optionally be passed as a parameter from one context to another. In such case, a context copy of the business object may be created which includes a duplicate of the original data and excludes a context variable. As another option, an exception may be thrown when an attempt is made to create a copy of a business object being altered by one logical unit of work for another logical unit of work.

Detailed Description Text (2895):
The business object may also be sent to another context as at least one of a single focus of a window that is being created and a parameter in an explicit

parameter-passing mechanism. Additionally, the copies of the business objects may be created from a same retrieved data stream. As a further option, receiving a request to make changes to a copy of the business object of one of the logical units of work and changing that copy of the business object may further include the broadcasting of the change to the other logical units of work.

Detailed Description Text (2913):
The aforementioned telecommunications example had two separate business LUWs for the account payment and account services functions. Although both activities may be related by the same logical account, this pattern gives each a different context copy. Then, when the customer representative cancels the addition of call waiting, she can still save the payment details.

Detailed Description Text (2919):
The following implementation assumes that the L UW Context pattern is used to help separate the LUWs.

Detailed Description Text (2920):
Each instance of a business object knows which LUW owns it. That is, each instance knows its context. By definition, context gives something a scope, a frame of reference, a relationship to other things. To provide this relationship, an actual LUW Context object will hold onto business objects which share a business LUW.

Detailed Description Text (2921):
In addition, each business object can point to its context. In that manner, business objects know their LUW. This could be useful, for example, while building a domain model. Then, the parent object could propagate its context to a linked, child object.

Detailed Description Text (2922):
Business objects owned by the same business LUW share the same LUW Context, whereas different LUWs have different contexts. Each context therefore contains its own "working-storage" copy of the model. This delineates an individual workspace, or scratchpad, for each LUW.

Detailed Description Text (2923):
At a higher level, each activity object which represents a business LUW has its own context object. That context remains with the activity throughout its entire lifecycle. For initialization, creating a new LUW activity also creates a new context instance for that activity. This context will then be passed downwards, to all business objects, as part of navigation.

Detailed Description Text (2924):
Eventually, when the activity closes, it releases its LUW Context. This correspondingly releases all business objects. They can then be garbage collected, because the only LUW using those objects just closed. A context's lifecycle therefore corresponds directly to its activity's lifecycle.

Detailed Description Text (2925):
Preserving Context Boundaries

Detailed Description Text (2926):
Every context has a scope which limits the business LUW. This context boundary cannot be violated with objects from other LUW Contexts. For, the same instance of a business object cannot live in two different contexts. Otherwise, changes made to that instance would affect two different LUWs.

Detailed Description Text (2927):
It is often necessary, however, to pass a business object as a parameter from one context to another. For example, a user may open up a customer details window based on a selection from a search window. The selected customer becomes the focus of the new window, but it was instantiated in the search context. It is the responsibility of the details window to take the passed-in customer and make a context copy of it. A context copy duplicates the original's data, excluding the context variable, which is re-set to the new context. The copied customer can then be safely used and

modified within the details context.

Detailed Description Text (2928):
A business object can be passed as parameter to another context as: the single focus of a window that is being created a parameter in an explicit parameter-passing mechanism

Detailed Description Text (2930):
MeterMaintenanceActivity::prorateMeterRead(MeterRead aMeterRead) { // Creates a new activity to prorate <aMeterRead>. This will manually adjust the // read charges, based on corrections from the location, the office, etc. // Pseudo-code below. // Create the new activity instance by reflection, based on the class name, and // give it a new context and <aMeterRead> as focus. newProrateActivity=this.newActivity(this.prorateMeterReadClassName( ), aMeterRead); // Other initialisation here . . . newProrateActivity.startup( ); }

Detailed Description Text (2931):
The newActivity( ) architecture method instantiates a new activity, instantiates a new context, and creates a context copy of aMeterRead that the new activity can use.

Detailed Description Text (2932):
Sometimes an activity cannot get enough information simply by navigating from the focus. Non-focus information that must be passed as an additional parameter could be handled in the following manner:
MeterMaintenanceActivity::prorateMeterReadWithCorrection(MeterRead originalMeterRead, MeterRead correctedMeterRead) { // Creates a new activity to prorate <originalMeterRead> based on measurements // in <correctedMeterRead>. Pseudo-code below. (Duplicates some code above // for clarification.) // Create the new activity instance by reflection, based on the class name, and // give it a new context and <originalMeterRead> as focus. newProrateActivity=this.newActivity(this.prorateMeterReadClassName( ), originalMeterRead); // Pass along the corrected read, as well. This will create a context copy and // then use reflection to call the right public setter on the activity.
newProrateActivity.receive(aCorrectedMeterRead, "setCorrectedRead") ; // Other initialisation here . . . newProrateActivity.startup( ); }

Detailed Description Text (2933):
Here, the receive( ) framework method allows any business object to be passed across the context boundary. The receiving activity will automatically create a context copy and then call the specified setter method, with the copy as argument. The setter is application-specific, and it allows the activity to handle and store the context copy wherever it wants.

Detailed Description Text (2934):
FIG. 195 illustrates the Context Copying Protects Context Boundanes.

Detailed Description Text (2935):
A dirty object should not be safely copied into a new LUW context. Otherwise, the second LUW would begin using information that was half-completed in the first LUW. Again, this violates the isolation requirement. The second LUW could save its changes before the first LUW. This means the first LUW couldn't undo any changes it had made to the dirty object. Instead, to avoid this problem, an exception should be thrown when trying to copy dirty objects across contexts. This disallows users from beginning a new LUW based on half-entered data.

Detailed Description Text (2936):
Thus, context copying allows LUW contexts to share parameter information while preserving context boundaries.

Detailed Description Text (2938):
Although LUW contexts manipulate separate copies of business objects, they can often share the same retrieved data stream. For example, when a workstation retrieves data for Customer ABCD, the returned stream can be stored in a global cache. If another context wants to later instantiate its own copy of Customer ABCD, it can reuse the

details stored in the stream cache. This improves performance, by avoiding a redundant request to the remote data store.

Detailed Description Text (2939):
Context "Refresh"

Detailed Description Text (2940):
Each LUW, while working on its data, is independent of the other LUWs. From that perspective, each LUW context manipulates data that, to its knowledge, is the most current information from the data store. One instance's changes remain invisible to another copy of the same business entity, during the course of normal processing.

Detailed Description Text (2941):
However, when an LUW context successfully commits changes, it will have more current data than other contexts which it intersects. This up-to-date data can be broadcast and shared with the other contexts. These contexts can then decide to transparently incorporate the changes or not.

Detailed Description Paragraph Table (13):
Object retrieved Object retrieved has has not changed changed since read since read Object in cache has Raise an error. At Ignore the changed since read commit transaction retrieved object, there will be an the object in cache optimistic lock failure is newer and the so it is better to raise changes should not it now. be lost. Object in cache has not Replace the object in Ignore the changed since read cache with the object retrieved object, it retrieved since the is ready in the retrieved object is cache. newer.

CLAIMS:

1. A method for detecting an orphaned server context, comprising: (a) maintaining a collection of outstanding server objects; (b) creating a list of contexts for each of the outstanding server objects; (c) adding to the list a compilation of clients who are interested in each of the outstanding server objects; (d) recording on the list a duration of time since the clients invoked a method accessing each of the contexts of the outstanding server objects; (e) examining the list at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed; (f) selecting contexts that have not been accessed in the predetermined amount of time; and (g) sending information to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

2. A method as recited in claim 1, further comprising waiting a preselected amount of time for receiving a response from one of the clients, wherein the context is deleted if a response from one of the clients is not received within the predetermined amount of time.

3. A method as recited in claim 1, further comprising receiving a response from one of the clients requesting that one of the contexts be maintained.

4. A method as recited in claim 3, further comprising updating on the list a time the context was last updated to a current time upon receipt of the response.

7. A computer program embodied on a computer readable medium for detecting an orphaned server context, comprising: (a) a code segment that maintains a collection of outstanding server objects; (b) a code segment that creates a list of contexts for each of the outstanding server objects; (c) a code segment that adds to the list a compilation of clients who are interested in each of the outstanding server objects; (d) a code segment that records on the list a duration of time since the clients invoked a computer program accessing each of the contexts of the outstanding server objects; (e) a code segment that examines the list at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed; (f) a code segment that selects contexts that have not been accessed in the predetermined amount of time; and (g) a code segment that sends information to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

8. A computer program as recited in claim 7, further comprising a code segment that waits a preselected amount of time for receiving a response from one of the clients, wherein the context is deleted if a response from one of the clients is not received within the predetermined amount of time.

9. A computer program as recited in claim 7, further comprising a code segment that receives a response from one of the clients requesting that one of the contexts be maintained.

10. A computer program as recited in claim 9, further comprising a code segment that updates on the list a time the context was last updated to a current time upon receipt of the response.

13. A system for detecting an orphaned server context, comprising: (a) software that maintains a collection of outstanding server objects; (b) software that creates a list of contexts for each of the outstanding server objects; (c) software that adds to the list a compilation of clients who are interested in each of the outstanding server objects; (d) software that records on the list a duration of time since the clients invoked a system accessing each of the contexts of the outstanding server objects; (e) software that examines the list at predetermined intervals for determining whether a predetermined amount of time has passed since each of the objects has been accessed; (f) software that selects contexts that have not been accessed in the predetermined amount of time; and (g) software that sends information to the clients identifying the contexts that have not been accessed in the predetermined amount of time.

14. A system as recited in claim 13, further comprising software that waits a preselected amount of time for receiving a response from one of the clients, wherein the context is deleted if a response from one of the clients is not received within the predetermined amount of time.

15. A system as recited in claim 13, further comprising software that receives a response from one of the clients requesting that one of the contexts be maintained.

16. A system as recited in claim 15, further comprising software that updates on the list a time the context was last updated to a current time upon receipt of the response.

19. A method for cleaning-up orphaned server contexts, comprising: (a) providing a plurality of objects in an object-oriented computer system; (b) providing a plurality of server contexts for use by a plurality of processes; (c) creating a server object context for each of the plurality of objects as each object comes into use in one of the plurality of server contexts; and (d) identifying for clean-up one of the plurality of server object contexts in one of the server contexts after said server object context is not accessed for a predetermined period of time.